Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Informática

Tiago Alexandre da Costa Gonçalves

Implementação e testes de robustez do protocolo tempo-real FTT-CAN

Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Informática

Tiago Alexandre da Costa Gonçalves

Implementação e testes de robustez do protocolo tempo-real FTT-CAN

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Paulo Bacelar Reis Pedreiras do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Doutor Valter Filipe Miranda Castelão da Silva da Escola Superior de Tecnologia e Gestão de Águeda

Dedico este trabalho aos meus pais, restantes familiares, namorada e amigos.

o júri

presidente	Prof. Doutor António Manuel de Brito Ferrari Almeida Professor Catedrático do Dep. de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
arguente	Prof. Doutor Nuno Miguel Morais Rodrigues Departamento de Engenharia Electrotécnica da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria
orientador	Prof. Doutor Paulo Bacelar Reis Pedreiras Professor Auxiliar do Dep. de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
co-orientador	Prof. Doutor Valter Filipe Miranda Castelão da Silva Professor Adjunto, Escola Superior de Tecnologia e Gestão de Águeda, Universidade de Aveiro

agradecimentos

A escrita desta Dissertação, exigiu sem dúvida um enorme esforço e a todos aqueles que me acompanharam nesta jornada o meu profundo agradecimento.
Aos meus orientadores, Professor Paulo Pedreiras e Professor Valter Silva, pela ajuda, os erros apontados e o esforço dispendido que foram cruciais na resolução dos problemas com que me deparei. A sua amizade e compreensão contribuíram sem dúvida para o sucesso deste trabalho.
Um agradecimento muito especial aos meus pais, que me ofereceram a possibilidade de realizar esta escrita e pela incondicional motivação, compreensão, força e confiança necessárias para a realização dos meus sonhos.
À Sofia, minha namorada, que sempre me apoiou e acompanhou todo o meu trabalho ao longo de toda esta Dissertação. Pelas alegrias, angústias e essencialmente pela compreensão.
Um agradecimento especial ao Artur Melo e ao Daniel Carvalho, pela sua amizade incondicional, as palavras encorajadoras e as "conversas de Tasca" que sempre me animam.
Aos meus colegas e amigos da PrataGás e Franklim Prata, pela compreensão, motivação e em especial pela amizade demonstradas.
Aos meus amigos e colegas pela discussão de ideias e sugestões.
A todos que de um modo directo ou indirecto contribuíram e contribuem para o meu desenvolvimento profissional e pessoal.

Obrigado a todos!

palavras-chave

Comunicações tempo-real, Sistemas distribuídos, Redundância, FTT-CAN-*Flexible Time-Triggered on CAN*, CAN-*Controller Area Network*.

resumo

A crescente quantidade e complexidade dos sensores utilizados em muitas aplicações distribuídas tem motivado a necessidade de desenvolver protocolos de comunicação que permitam maiores taxas de transmissão de dados. Contudo, muitas dessas aplicações possuem requisitos específicos, como previsibilidade e tolerância a falhas, que têm que ser satisfeitas.

Recentemente foi desenvolvida uma melhoria ao protocolo FTT-CAN, criado anteriormente na Universidade de Aveiro, que adiciona capacidades de tolerância a falhas ao protocolo base e ao mesmo tempo aumenta a largura de banda do tráfego com menor importância. Estas funcionalidades são baseadas em barramento replicado e controladores CAN. Mensagens críticas são transmitidas em ambos os barramentos, enquanto que mensagens não críticas são transmitidas num único barramento.

O trabalho realizado no âmbito desta dissertação consiste em portar a implementação do *Slave* da versão original da *stack* do protocolo FTT-CAN, não replicada e baseada no microcontrolador PIC18F258 para o microcontrolador dsPIC30F6012A, com dois controladores CAN. Um conjunto de testes experimentais está também incluído, de forma a avaliar a correcção lógica da implementação assim como o desempenho em termos de latência, *jitter* e *overhead*. Por fim, são também apresentados alguns testes de compatibilidade com a implementação de referência em PIC18F258.

keywords

Real-Time Communications, Distributed Systems, Redundancy, FTT-CAN-*Flexible Time-Triggered on CAN*, CAN-*Controller Area Network*.

Abstract

The increasing number and complexity of the sensors used in many distributed applications is fostering the need to develop network protocols that allow higher transmission rates. However, many of these applications have specific requirements, such as predictability and fault-tolerance, which have to be satisfied.

Recently it was developed an enhancement to the FTT-CAN protocol, developed earlier at the University of Aveiro, that adds fault-tolerance features to the base protocol whilst, at the same time, increases the available bandwidth to the generic traffic. These features are based on a replicated bus and CAN controllers. Critical messages are transmitted in both buses, while non-critical messages are transmitted in a single bus.
The work carried out in this dissertation consists in porting the Slave part of the FTT-CAN protocol stack from the original, non-replicated, implementation based on the PIC18f258 micro-controller to the dsPIC30F6012A dual-CAN micro-controller. A set of experimental tests is also included, in order to assess the logical correctness of the implementation as well as to have performance figures regarding latency, jitter and overhead. Finally, compatibility tests with the PIC18f258 reference implementation are also presented.

Índice

Índice de Figuras

De modo semelhante ao que sucedeu no cenário e), no cenário f) a latência devido ao processamento da TM aumentou para cerca de 95 µs. O instante de início de transmissão

Índice de Tabelas

Lista de Siglas e Acrónimos

CAN – *Controller Area Network*

CD – *Collision Detection*

CNI – *Controller Network Interface*

CPU – *Central Processing Unit*

CRC – *Cyclic Redundant Check*

CSMA/CD – *Carrier Sense Multiple Access with Collision Detection*

CSMA – *Carrier Sense Multiple Access*

DSP – *Digital Signal Processor*

EC – *Elementary Cycle*

FTT-CAN – *Flexible Time-Triggered on CAN*

ISO – *International Organization for Standardization*

OSI – *Open Systems Interconnection*

RTR – *Remote Transmission Request*

SED – *Sistemas Embutidos Distribuídos*

SR – *Status Register*

SRDB – *System Requirements Data Base*

SRT – *Synchronous Requirements Table*

SRU – *Smallest Replaceable Unit*

TDMA – *Time Division Multiple Access*

TM – *Trigger Message*

TTP/C – *Time Triggered Protocol / class C*

Capítulo 1

Introdução

1.1. Enquadramento

Actualmente, os Sistemas Embutidos Distribuídos (SED) encontram-se por toda a parte e mesmo sem nos apercebermos, eles tornam a nossa vida bastante mais fácil e cómoda. Um SED pode ser definido como um conjunto de nodos dispersos que se encontram interligados de alguma forma e onde a cada nodo é atribuído um conjunto de operações a realizar, cuja quantidade e complexidade poderão variar dependendo da aplicação e das opções do projectista do sistema. A interligação entre nodos é feita por meio de uma rede de comunicação, cujo objectivo é possibilitar a troca de informação entre nodos, para que trabalhem "em equipa" como se de um único se tratasse. Sistemas que operam desta forma podem ser facilmente encontrados em áreas como indústria, sistemas automóveis, aviónica e robótica.

A troca de informação entre tarefas que executam em nodos diferentes obriga a considerar o impacto da rede de comunicação, pois numa rede com baixa largura de banda, as tarefas locais podem executar muito mais rapidamente do que a velocidade a que a informação circula pela rede, o que pode levar ao consumo de dados que já não são considerados válidos no instante em que são recebidos.

Considere-se por exemplo, o caso de um sistema de controlo em malha fechada utilizando uma arquitectura distribuída. Se a rede se encontrar sobrecarregada, é provável que venha a ser introduzido um atraso não desprezável no envio/recepção dos dados e estes poderão chegar após a execução do controlador, originando um atraso de um período na malha, degradando consideravelmente a qualidade do controlo realizado o que pode levar a situações indesejáveis.

Em sistemas do tipo *safety-critical*, é imperativo garantir que a informação não só atinja o destino mas também que esse acontecimento se realize no instante correcto.

Uma forma de garantir que a informação chega no instante correcto, é através da utilização de uma técnica de escalonamento, que tem por objectivo definir uma determinada prioridade no envio da informação de acordo com as opções do projectista. Por outro lado, para garantir que a informação chega efectivamente ao destino, são utilizadas técnicas que promovem um envio redundante da informação, como é o caso da replicação de nodos e barramentos, oferecendo alternativas em caso de ocorrência de erro.

Hoje em dia existe já uma grande variedade de protocolos que incluem mecanismos de tolerância a falhas. Protocolos como o FlexRay [1] e o TTP/C [2] oferecem a possibilidade de utilizar mais do que um barramento como forma de implementar um envio redundante, desta forma a existência de falha de um dos barramentos não irá prejudicar o funcionamento do sistema, apenas o seu desempenho. Outros protocolos como o CANCentrate [3] e o ReCANCentrate [3] implementam mecanismos mais direccionados a tratar outros tipos de problemas como a falha de nodos, isolando o nodo problemático do resto do sistema.

A utilização de barramentos replicados não serve apenas para implementar redundância uma vez que a utilização paralela dos barramentos oferece também a possibilidade de aumentar a taxa de transmissão enviando dados diferentes por cada barramento, o que por sua vez representa um aumento da capacidade total de transmissão de informação por unidade de tempo.

O trabalho realizado no âmbito desta dissertação vem complementar aquele que foi desenvolvido em [4] e assenta na gestão flexível de largura de banda e redundância. Nesse trabalho foi implementada a replicação do *Master* do FTT-CAN [5], com nodos de *Backup*, com o objectivo de tomar o seu lugar na situação de ocorrência de erro do *Master* activo.

1.2. Objectivo da dissertação

Actualmente, uma das aplicações do protocolo FTT-CAN é no campo da robótica [5], onde por vezes são utilizados sensores que requerem elevada largura de banda, como sonares e câmaras. Sensores deste tipo facilmente ocupam a largura de banda disponível

deixando muito pouco para os restantes componentes. Deste modo foi proposto em [6] uma nova implementação do protocolo FTT-CAN de forma a combinar redundância com o aumento da largura de banda.

O trabalho realizado no âmbito desta dissertação teve como objectivo a implementação dos *Slaves* do protocolo FTT-CAN, num sistema distribuído em que os nodos são baseados em dsPIC30F6012A, de forma a utilizar barramentos replicados para aumentar a largura de banda disponível ou promover o envio redundante de mensagens de extrema importância.

O trabalho foi dividido em três partes, em que a primeira fase consistiu em adaptar o código existente para PIC18F258 para funcionar em dsPIC30F6012A, utilizando apenas um dos dois controladores CAN. A segunda fase teve como objectivo a implementação dos módulos de gestão do segundo barramento, criando as rotinas necessárias ao seu funcionamento. A terceira fase assenta na necessidade de apresentar garantias de funcionamento e caracterizar devidamente o funcionamento do sistema e respectivo desempenho, deste modo esta parte foi dedicada à execução de testes extensivos do protocolo implementado nesta plataforma.

1.3. Estrutura da dissertação

Esta dissertação encontra-se estruturada em 5 capítulos, os quais se dividem em subcapítulos de acordo com os assuntos abordados.

Capítulo 2 – tem como objectivo abordar o estado de arte dos protocolos que implementam as redes de comunicação em tempo real. Aqui serão apresentados e explorados alguns dos protocolos que existem e são utilizados actualmente.

Capítulo 3 – visa descrever o trabalho realizado no âmbito desta dissertação, relatar de forma completa os procedimentos utilizados e justificar as escolhas tomadas.

Capítulo 4 – aqui é apresentado um conjunto de ensaios experimentais, as motivações que levaram à escolha dos testes realizados e os respectivos resultados.

Capítulo 5 – contém as conclusões obtidas do trabalho realizado, assim como algumas propostas de implementação futura para pontos que ainda poderão ser melhorados.

Capítulo 2

Protocolos de comunicação em sistemas distribuídos

2.1. CAN

Os protocolos existentes na década de 80 (I2C, D2B) possuíam fracas capacidades para aplicações de tempo-real. A ausência de suporte para comunicação *multi-Master* e os fracos mecanismos de detecção e gestão de erros desses protocolos levaram à necessidade de criar uma nova solução [7].

É com o objectivo de suprir esta necessidade que a Bosch desenvolve o CAN (Controller Area Network) nos finais dos anos 80. O CAN foi lançado como um protocolo para aplicações automóveis, com o objectivo de aumentar a eficiência dos sistemas automóveis, tornando-os mais robustos e seguros ao mesmo tempo que se reduzia o número de cablagens no seu interior. A figura 2.1 ilustra a arquitectura geral do CAN.

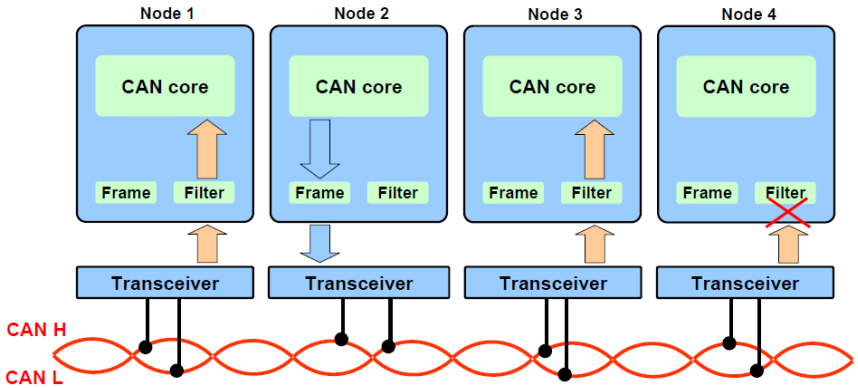

Figura 2.1 – Exemplo da arquitectura utilizada no CAN, retirado de [7].

Foram registados dois *standards* para a camada física deste protocolo, o ISO 11898-2, para aplicações de alta velocidade, permitindo taxas de transmissão até 1 Mb/s, mas onde o tamanho do barramento se encontra limitado a 40m e o *standard* ISO 11898-

3, para aplicações de baixa velocidade com taxas até 125 Kb/s com uma limitação de 32 nodos ligados na mesma rede. A figura 2.2 ilustra onde estes dois *standards* se encaixam no modelo OSI.

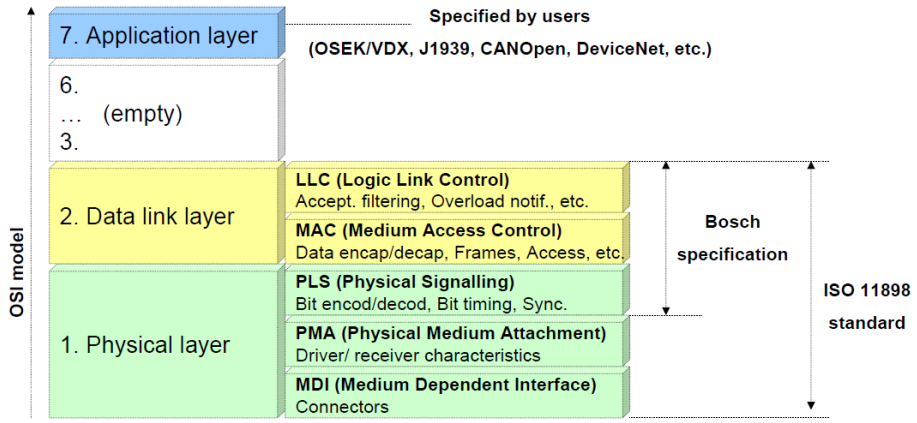

Figura 2.2 – Especificação Bosch para o CAN de acordo com o modelo OSI, retirado de [7].

O CAN é um protocolo de comunicação série assíncrono que se baseia no conceito de *multi-Master*, isto é, quando o barramento se encontra livre qualquer nodo pode iniciar a transmissão. Desta forma, utiliza o CSMA/CD (*Carrier Sense Multiple Access with Colision Detection*) em que o CSMA significa que qualquer nodo deve escutar o meio durante um período de inactividade antes de iniciar a transmissão. O CD (*Collision Detection*) significa a presença de um mecanismo para detectar e resolver colisões, no caso do CAN a arbitragem é feita *bit* a *bit* e de uma forma não destrutiva, o que significa que a mensagem de maior prioridade não é afectada mesmo quando é detectada uma colisão.

A arbitragem tem em conta a definição de *bits* recessivo/dominante e é feita utilizando o campo *Message Identifier* da mensagem, assim torna-se determinística e quanto menor o valor do identificador maior a prioridade, uma vez que o *bit* dominante é o zero.

No CAN, as mensagens não são enviadas directamente de nodo para nodo, pois não existem mecanismos de endereçamento de nodos mas sim das próprias mensagens, sendo estas enviadas em *broadcast* e recebidas por todos os nodos, cabendo a responsabilidade de filtrar devidamente a mensagem aos nodos, sendo-lhes possível

descartar a mensagem recebida ou armazená-la para processamento. Dentro deste protocolo existem quatro tipos de mensagens a que é possível recorrer.

Data Frame – Este é o tipo de mensagem vulgarmente utilizado e permite o transporte de informação entre nodos. Os campos mais importantes que a constituem são o identificador, utilizado para os processos de arbitragem e que pode ter 11 ou 29 bits, o campo de dados que contem a informação a enviar e o CRC utilizado para verificação de erros. Possui um total de sete campos.

Remote Frame – Esta mensagem tem como função solicitar a transmissão de outra mensagem. Em termos estruturais assemelha-se com a *Data Frame* mas não possui campo de dados e é identificada como *Remote Frame* enviando o *bit* RTR como recessivo.

Error Frame – Quando um nodo detecta um erro, este envia uma *Error Frame* de modo a sinalizar o erro. Uma vez que este tipo de mensagem viola as regras de construção de *frames* CAN, os nodos restantes detectam a condição de erro e dão inicio também ao envio de *Error Frame*s.

Overload Frame – A estrutura deste tipo de mensagem é semelhante à *Error Frame* e é utilizado quando um nodo se encontra tão ocupado que necessita que as mensagens a ele destinadas sejam atrasadas. Cada nodo apenas pode enviar duas *Overload Frames* consecutivas.

É utilizado ainda um poderoso mecanismo de detecção de erros que consiste num CRC com 15 *bits*, em que quando a verificação do CRC falha é assumido um erro e iniciada a retransmissão da mensagem. A figura 2.3 apresenta o diagrama de estados relativo ao tratamento de erros do CAN, as transições possíveis e respectivas condições de transição.

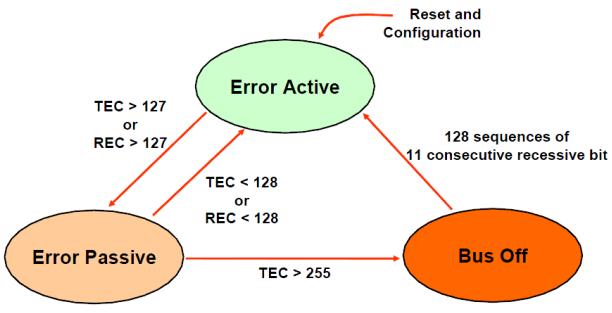

Figura 2.3 – Diagrama de transição de estados para tratamento de erros retirado de [7].

O CAN possui ainda outras capacidades de tempo real como a sua transmissão com baixa latência e taxas de transmissão até 1Mbps [7]. Assim permite um excelente controlo sobre os sistemas distribuídos de tempo-real que nele se baseiam, trocando dados rapidamente. A sua popularidade foi crescendo ao longo dos anos à medida que as suas potencialidades eram exploradas e hoje em dia o CAN faz parte de sistemas de alta fiabilidade, como por exemplo, sistemas médicos e industriais.

2.2. FTT-CAN

O FTT-CAN é um protocolo de comunicação que usa o CAN como camada MAC e foi desenvolvido com o objectivo de combinar um elevado nível de flexibilidade operacional com garantias temporais [6].

O protocolo é gerido por um nodo especial designado por *Master*. Este envia uma mensagem especial, a *Trigger Message* (TM) de forma a sincronizar o sistema e informar os restantes nodos de quais as mensagens síncronas com permissão de envio durante o EC actual e qual o tamanho da janela síncrona. O *Master* é ainda responsável pela admissão e agenda *on-line* de mensagens de uma forma controlada e flexível, oferecendo garantias de pontualidade [6]. A figura 2.4 ilustra a arquitectura geral do protocolo FTT-CAN, onde se identifica um nodo *Master* e vários *Slaves*.

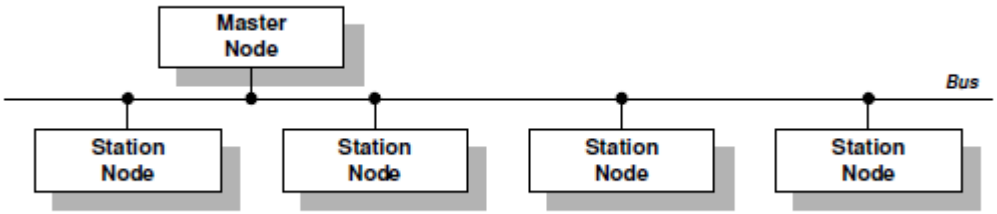

Figura 2.4 – Arquitectura geral do FTT-CAN, retirado de [6].

O período entre a recepção de duas TMs consecutivas define a duração do *Elementary Cycle* (EC). Este divide-se em duas janelas temporais, a janela síncrona e a janela assíncrona. A janela síncrona é gerada dinamicamente de acordo com o número de

mensagens periódicas previstas para o EC, deste modo a sua duração irá variar com a quantidade de tráfego agendado para o EC que decorre. O envio de mensagens periódicas apenas é permitido dentro desta janela e de modo semelhante, a transmissão de mensagens assíncronas será realizada apenas durante a janela assíncrona.

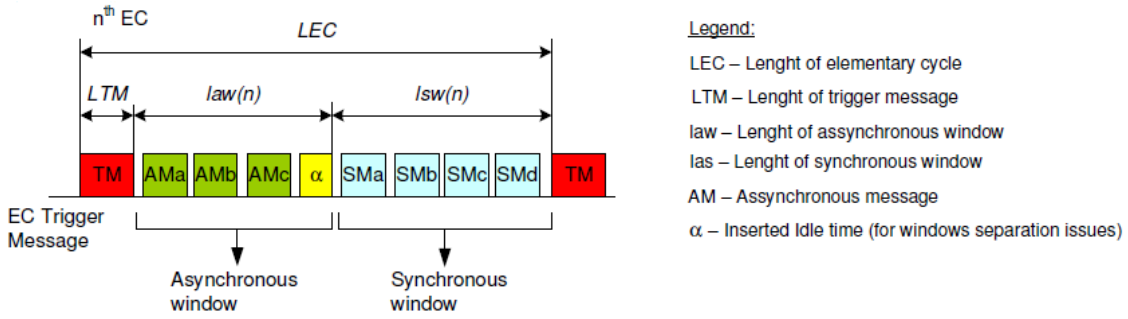

Figura 2.5 – Ciclo elementar do FTT-CAN, retirado de [6].

Neste protocolo, a responsabilidade de resolver as colisões detectadas fica a cargo da arbitragem nativa do CAN. Deste modo e de forma a cumprir o agendamento previsto para o EC, as mensagens síncronas que sejam transmitidas incorrectamente não são retransmitidas no EC que decorre e em caso de retransmissão, esta deverá ocorrer num EC posterior.

O *Master* contém ainda uma Base de Dados de Requisitos do Sistema (SRDB) da qual faz parte uma tabela denominada Tabela de Requisitos Síncronos (SRT) que guarda a descrição dos fluxos de mensagens periódicas [6] e é com base nesta informação que o escalonador constrói a agenda das mensagens síncronas para cada EC.

Esta informação é codificada na TM e enviada aos *Slaves*, que a descodificam e utilizam para saber se podem dar início à respectiva transmissão durante a próxima janela síncrona, o que só acontecerá no caso de o nodo ser produtor da mensagem em questão.

O agendamento das mensagens é feito *on-line* e qualquer alteração realizada à SRT em tempo de execução irá reflectir-se no tráfego do barramento, resultando num comportamento bastante flexível. Isto é conseguido através de mensagens de controlo, enviadas dos *Slaves* para o *Master*, com as informações das variáveis a adicionar, modificar ou remover. Infelizmente a utilização destas mensagens introduz *overhead* na rede deixando menos espaço para transmissão de dados, embora o impacto seja residual,

uma vez que apenas é preciso uma mensagem assíncrona e em casos normais não é necessário enviar grande quantidade de mensagens de controlo.

Relativamente a mecanismos de tolerância a falhas, o FTT-CAN possui uma forma de resolver a falha do nodo *Master* que consiste na utilização de um ou mais *Masters* de *Backup* que estão sincronizados com os ECs e "escutam" o barramento aguardando a recepção da TM, sempre que for detectada a sua ausência no barramento, um *Master Backup* passará ao estado activo e tomará o lugar do *Master* uma vez que este teve erro, evitando a falha de todo o sistema. Contudo, este protocolo ainda não contempla a falha do barramento, portanto se este falhar todo o sistema poderá ser afectado.

2.3. CANCentrate e ReCANCentrate [3]

O protocolo CANCentrate resulta da procura por sistemas baseados em CAN mas que apresentem um nível de fiabilidade superior, pois num sistema CAN convencional uma falha num dos componentes é suficiente para que todo o sistema falhe.

Este protocolo utiliza uma arquitectura em estrela, utilizando um *hub* com várias portas e onde cada nodo se liga a uma porta, concentrando todas as ligações no *hub*, como se pode verificar na figura seguinte.

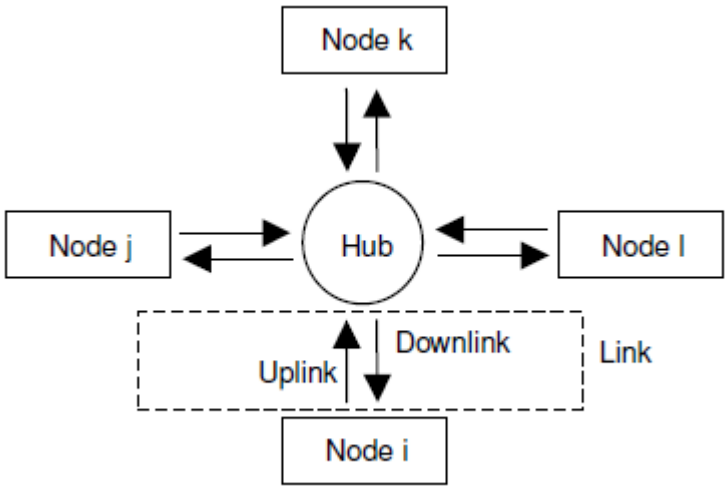

Figura 2.6 – Exemplo da arquitectura CANCentrate, retirado de [3].

A comunicação é realizada por meio de duas ligações designadas de *uplink* e *downlink*, em que a informação flui do nodo para o *hub* e do *hub* para o nodo, respectivamente.

Numa topologia baseada no CAN convencional, uma falha pode dar origem a uma grave perda de comunicação, enquanto que num sistema CANCentrate cada porta é definida como uma zona isolada de falhas, deste modo quando ocorre uma avaria num nodo ou em determinado segmento do barramento, o *hub* detectará a falha e irá conter a mesma naquela porta, isolando a falha localmente e evitando que todo o sistema seja afectado pela avaria, mantendo o normal funcionamento do sistema, desde que o afastamento do nodo problemático não prejudique o funcionamento dos restantes nodos.

Como referido na secção 2.1. uma das características do CAN é a transmissão segundo um modelo baseado em *bits* dominantes e recessivos. De forma a manter este tipo de transmissão o *hub* necessita de um mecanismo para implementar a função lógica AND de todas as contribuições das portas numa fracção da duração de um *bit* para não afectar a resposta que resulta ao receber uma mensagem.

Um dos objectivos do *hub* é implementar um mecanismo para detecção de portas que não funcionem correctamente e isolá-las do resto do sistema. De forma a detectar portas problemáticas é necessário diferenciar os sinais recebidos pelo *hub* daqueles que são enviados por *broadcast* para os nodos, sendo essa a razão de utilizar as duas ligações, *uplink* e de *downlink*.

O *hub* que implementa as capacidades de tolerância a falhas do CANCentrate é constituído por três módulos, o módulo de *Input/Output*, o módulo de *Coupler* e o módulo de *Fault-Treatment*.

O primeiro módulo é um interface que converte sinais físicos em lógicos e vice-versa, permitindo que o *hub* processe as mensagens recebidas e reenvie mensagens após processamento.

O segundo módulo implementa a função AND responsável por acoplar os diversos sinais de *uplink*, neste módulo são ainda implementadas várias funções de OR lógico

entre os sinais recebidos dos nodos e um sinal que permite activar ou desactivar a contribuição de canais de *uplink* específicos.

O terceiro e último módulo é responsável por detectar e isolar as portas em que se verifiquem erros.

Vale a pena salientar que os cabos utilizados de e para o *hub* são do mesmo tipo que os utilizados no CAN típico, fio de cobre entrançado que fornece uma boa resiliência quando exposto a interferências electromagnéticas.

Apesar das melhorias introduzidas pelo protocolo CANCentrate, especialmente em termos de detecção e tratamento de falhas, a fiabilidade oferecida pode não ser suficiente para suprir as exigências de determinados sistemas.

Uma vez que o *hub* se encontra na base da sua arquitectura, se este falhar, todo o sistema falha, o que é inaceitável em sistemas críticos. Assim, verificou-se a necessidade de implementar um sistema redundante, replicando o *hub* e as ligações *hub*-nodo. A solução proposta foi designada por ReCANCentrate e consiste no uso de dois *hubs* replicados, duplicando também o número de ligações aos nodos.

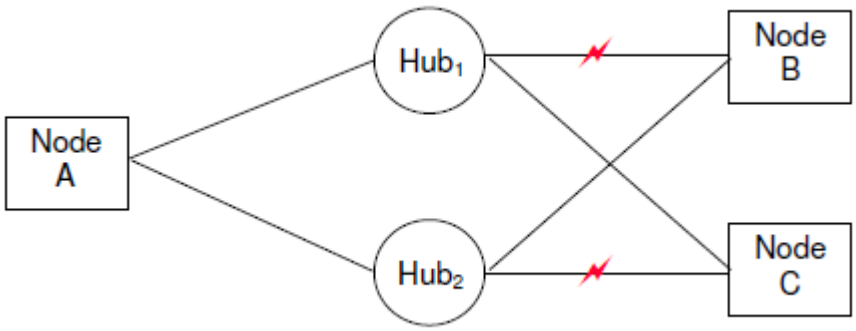

Figura 2.7 – Topologia em estrela utilizando redundância, retirado de [3].

A replicação do *hub* juntamente com as ligações permitem assim um caminho alternativo entre o nodo origem e o nodo destino, pois caso uma das ligações esteja interrompida por alguma razão, a informação seguirá pelo caminho alternativo exactamente da mesma forma, camuflando a avaria em termos de funcionalidade do sistema.

A transmissão e recepção são feitas da mesma forma que no CANCentrate utilizando o mesmo tipo de ligações de *uplink* e *downlink* cuja diferença está no uso de caminhos replicados.

Outra diferença fundamental na arquitectura do ReCANCentrate é a existência de ligações *interlink*, estabelecidas entre *hubs*. Estas ligações têm como objectivo partilhar as contribuições de cada nodo com o outro *hub* gerando assim um sinal único como se ambos os *hubs* fossem um só elemento.

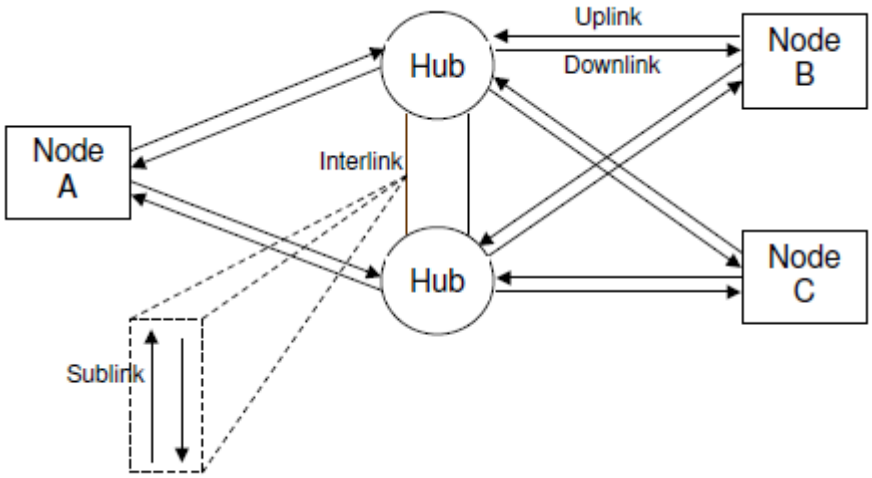

Figura 2.8 – Exemplo da arquitectura ReCANCentrate, retirado de [3].

Apesar das melhorias apresentadas com a solução ReCANCentrate, esta pode dar origem a alguns inconvenientes. Por exemplo, se não for garantido que as contribuições de um *hub* cheguem correctamente ao outro, ambos os *hubs* irão comportar-se de forma independente colocando em questão a funcionalidade pretendida. Numa situação deste género haverá grande probabilidade de ocorrer dessincronização entre mensagens e respectivos duplicados, apresentando-se como se de uma outra mensagem completamente diferente se tratasse. Outra consequência poderá ser o aparecimento de domínios isolados entre *hubs*.

Deve-se ter ainda em conta, que esta arquitectura impõe a utilização de ligações adicionais, quer entre *hubs* quer entre *hubs* e nodos. E no caso de ser possível utilizar mais que dois *hubs* replicados, o número de ligações crescerá exponencialmente com o número de *hubs* utilizados.

Tanto o CANCentrate como o ReCANCentrate são baseados em CAN e deste modo herdam todas as características do mesmo, como por exemplo os seus mecanismos de arbitragem e transmissão.

2.4. TTP/C [8]

O TTP/C, à semelhança do CAN e *FlexRay*, utiliza um conjunto de nodos (que no caso do TTP/C se designam por SRU, *smallest replaceable unit*). É um protocolo baseado em TDMA (*Time-division Multiple Access*), que consiste em atribuir a total largura de banda do barramento a um nodo em determinado instante e durante um tempo limitado, denominado por "SRU *slot*". Deste modo, o acesso ao barramento é previsível e livre de colisões, o que melhora consideravelmente o desempenho do sistema em geral. A figura seguinte exemplifica a forma como este acesso é feito.

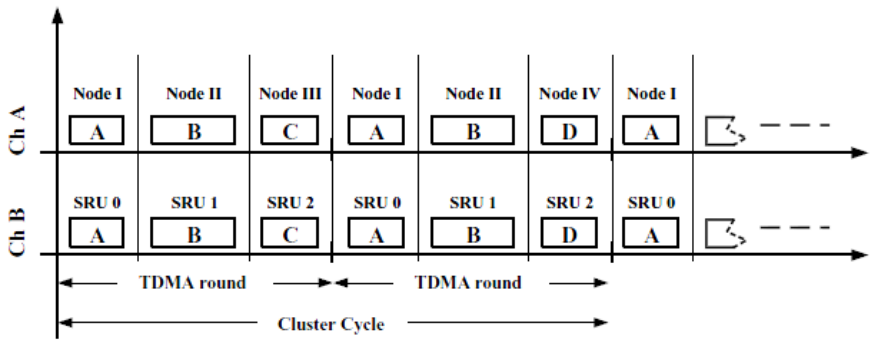

Figura 2.9 – Exemplo do funcionamento baseado em TDMA, retirado de [8].

O acesso ao barramento está dividido em ciclos (*Cluster Cycles*), que no contexto do TTP/C, estão definidos como o conjunto de "TDMA *rounds*" diferentes. Um "TDMA *round*" é uma sequência de "SRU *slots*" em que todos os nodos podem enviar no máximo duas tramas, uma por canal ou ambas no mesmo canal. Cada trama pode ter um comprimento máximo de 16 *bytes*.

Um membro SRU pode ser tanto real como virtual, sendo que no caso de ser real terá um SRU *slot* atribuído de forma estática, enquanto um membro virtual consiste num conjunto de SRUs que partilham o mesmo SRU slot, embora apenas um deles tenha permissão para transmitir dentro de um TDMA *round*.

O protocolo TTP/C oferece vários mecanismos para protecção contra falha, entre eles o serviço "*Membership*" que tem como objectivo informar com atraso mínimo todos os nodos que determinado nodo falhou. No TTP/C existe um campo denominado de node *membership* vector que contem um *bit* por cada nodo num cluster cycle, onde esse *bit* está a 1 se o nodo estiver a funcionar correctamente e a 0 caso contrário.

Outro mecanismo para tolerância a falhas é o "*fail-silence*" em que o nodo é desenvolvido de forma a realizar o envio correctamente ou a não enviar de todo.

O TPP/C possui ainda um *Bus Guardian*, que tem como função conectar ou desconectar o barramento do driver, garantindo o acesso apenas nos instantes respectivos.

Cada nodo é composto por três elementos, um *Host*, um *Controller Network Interface* (CNI) e um controlador de comunicações TTP/C, como pode ser consultado na figura 2.10.

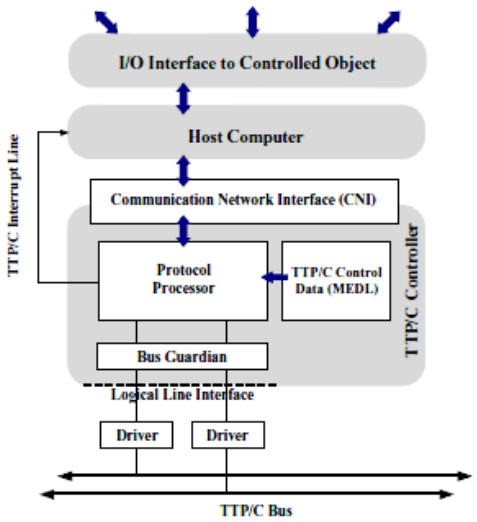

Figura 2.10 – Arquitectura interna de um SRU, retirado de [8].

O *Host* será onde a aplicação será carregada e executada, o *Controller Network Interface* tem a função de permitir ou negar que as aplicações possam aceder à rede de comunicação, o controlador de comunicações TTP/C permite a conectar o nodo ao barramento.

Outra característica importante é que este protocolo suporta sistemas baseados em barramentos replicados e promove ainda como meios de redundância adicionais a

replicação ao nível do hardware dos componentes *Host*, CNI e controlador TTP/C, ou seja, replicação de nodos, funcionando como se fossem apenas um.

2.5. *FlexRay* [10]

Com a crescente utilização de sistemas em número e em sofisticação no interior dos automóveis actuais, os protocolos utilizados deixaram de conseguir responder às exigências que foram emergindo, deste modo surge a necessidade por um protocolo com características mais adequadas à realidade actual.

Várias empresas decidem então criar o *FlexRay Consortium* de forma a investir na investigação e desenvolvimento de um novo protocolo que mais tarde foi denominado por *FlexRay* e cuja utilização já se começa a generalizar.

O *FlexRay* possui várias vantagens face ao seu concorrente directo, o CAN. Sendo as principais as taxas de transmissão, que podem atingir velocidades até 20x superiores e o facto de serem utilizados dois barramentos em vez de apenas um. Estas duas características permitem desde logo um maior potencial de transmissão de dados ao mesmo tempo que oferece uma solução redundante.

Outras vantagens são por exemplo o suporte a vários tipos de topologias, como barramento, estrela e híbridos. É também possível realizar a monitorização de agendamento por hardware e existe suporte tanto para dados síncronos como assíncronos.

Os nodos do *FlexRay* são um pouco mais complexos que os de outros protocolos já aqui falados e dividem-se em duas partes, a "*Controler Part*" e a "*Driver Part*", a parte do controlador consiste num processador e num controlador de comunicações, enquanto o driver possui um bus driver e um *bus guardian* por cada canal.

Ao contrário do CAN, no *Flexray* apenas existe um tipo de mensagem, a *data frame*.

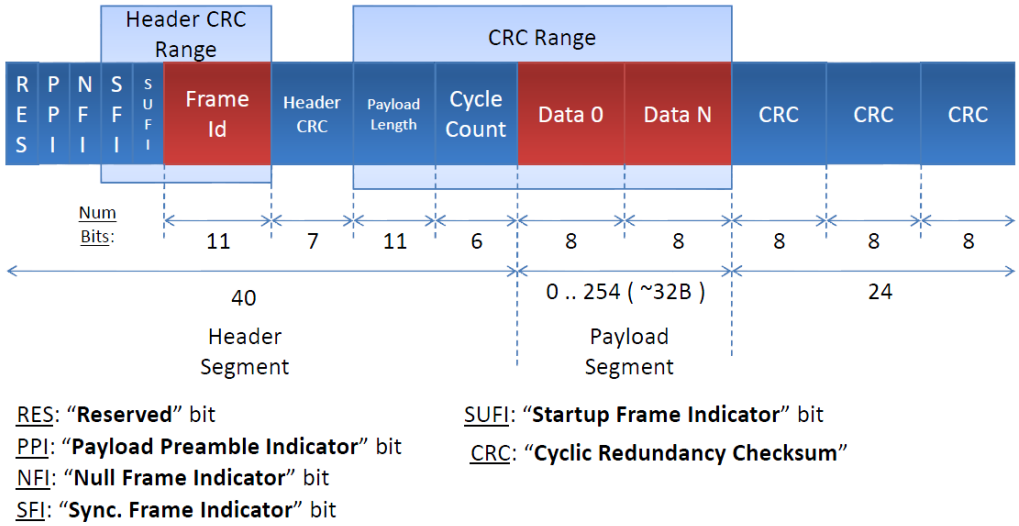

Figura 2.11 – *Data Frame* do *Flexray*, retirado de [10].

O *Flexray* cumpre as especificações exigidas para a sua utilização em aplicações *x-by-wire*, como o *steer-by-wire* e o *brake-by-wire*, sendo esse um dos objectivos apontados durante a especificação do protocolo.

As aplicações do tipo *safety-critical*, obrigam a que os sistemas de comunicação tempo-real em que se baseiam ofereçam garantias temporais e de fiabilidade de forma a minimizar a ocorrência de falhas catastróficas. Desse modo os protocolos referidos neste capítulo foram desenvolvidos com o objectivo de cumprir essas exigências. O trabalho realizado nesta dissertação visa complementar a implementação do protocolo FTT-CAN numa versão tolerante a falhas de barramento pelo que o capítulo seguinte será dedicado a descrever o procedimento realizado na implementação do FTT-CAN em DSP. Apresentando as alterações feitas, descrições do funcionamento dos vários módulos que fazem parte do FTT-CAN e apresentando os problemas encontrados e soluções respectivas.

Capítulo 3

Implementação de *stack* FTT-CAN para barramento replicado

3.1. Hardware

De forma a ser possível implementar o FTT-CAN na nova plataforma, foi necessário criar a nova infra-estrutura baseada em DSP. Uma vez que o desenho da placa em PCB já existia, apenas foi necessário realizar algumas alterações e correcções no desenho existente, como foi o caso da adição de uma porta série adicional e a correcção de algumas conexões trocadas. Toda a edição foi feita recorrendo ao programa de edição Eagle. Foi ainda necessário enviar o desenho das placas para produção e depois disso proceder à soldagem dos diversos componentes em cada uma das quatro placas utilizadas.

A placa inclui assim todos os elementos necessários para suportar a implementação do protocolo, como por exemplo, drivers de linha para CAN, conectores para facilitar as ligações de porta série, CAN e alimentação. A imagem 3.1 mostra o diagrama de blocos do sistema e a 3.2 o aspecto final da placa baseada no dsPIC30F6012A.

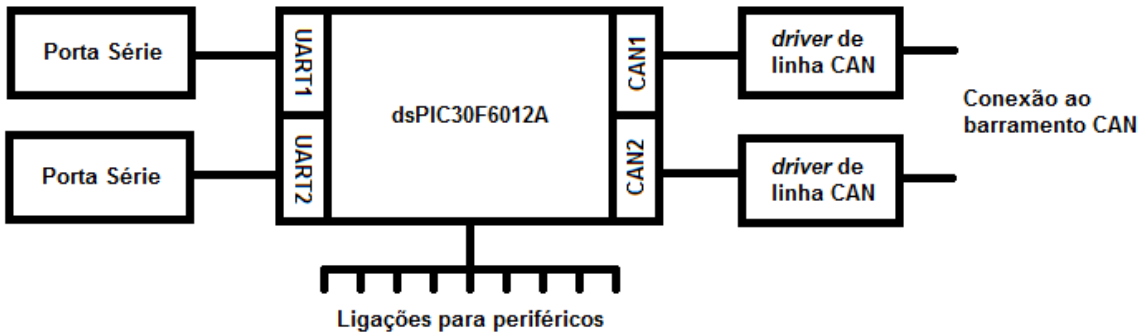

Figura 3.1 – Diagrama de blocos do sistema DSP.

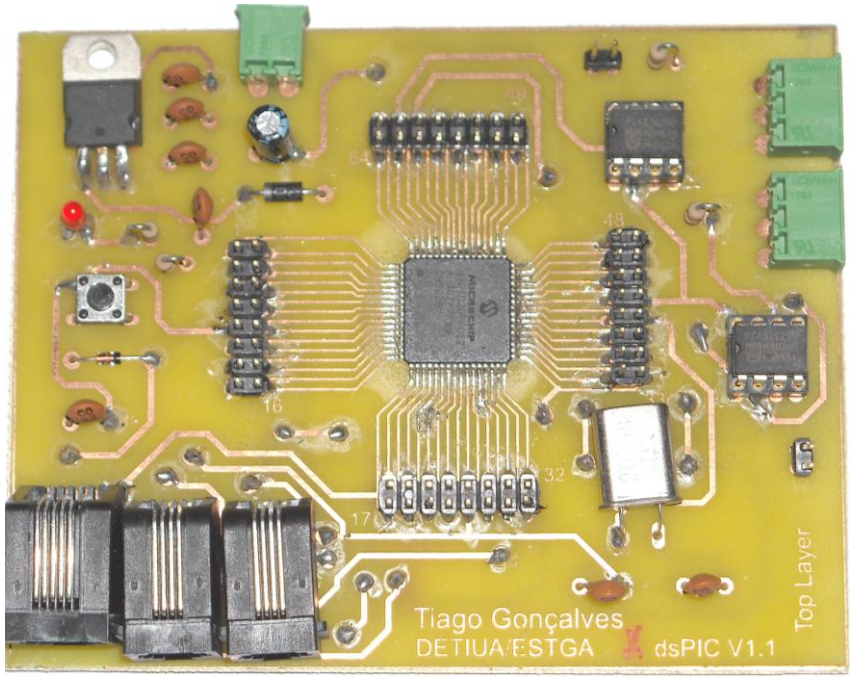

Figura 3.2 – Aspecto da placa dsPIC utilizada.

Como forma de comparação das duas infra-estruturas, a figura 3.3 ilustra a placa baseada em PIC18, utilizada anteriormente para implementar o FTT-CAN em barramento único.

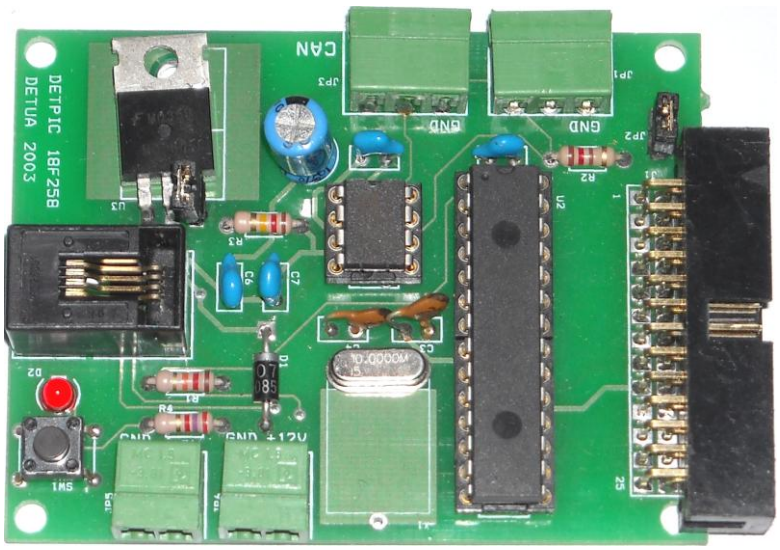

Figura 3.3 – Placa baseada em PIC18 utilizada para implementar o FTT-CAN em barramento único.

A placa utilizada neste trabalho ocupa uma maior área pois utiliza também um maior número de componentes, além disso, as placas do dsPIC ainda não foram alvo de

optimização da área utilizada, pois é algo que só fará sentido após implementação do protocolo na sua totalidade.

3.2. Mudança de plataformas

O sistema FTT-CAN em barramento simples encontra-se implementado com todas as suas funcionalidades num microcontrolador PIC18F258. Do ponto de vista do trabalho realizado nesta dissertação interessa referir que este microcontrolador possui como características uma operação de 8 *bits*, integra um controlador CAN e quatro *timers*.

A plataforma para a qual se pretende transitar baseia-se num dsPIC30F6012A, que no contexto da implementação do protocolo, possui como principais características a operação de 16 *bits*, os dois controladores CAN integrados e sete *timers*. A tabela 1 resume as diferenças referidas.

CPU	Arquitectura	Relógio(*)	# Barramentos CAN	# *Timers*
PIC18F258	8 *bits*	40 MHz	1	1 x 8-*bit*, 3 x 16-*bit*
dsPIC30F6012A	16 *bits*	80 MHz	2	5 x 16-*bit*, 2 x 32-*bit*

Tabela 3.1 – Apresenta as principais diferenças estruturais entre os PICs do ponto de vista do trabalho realizado. (*velocidades definidas na implementação do FTT-CAN)

Deste modo e utilizando por base o código fonte já existente para PIC18, a implementação começou pela alteração dos registos do processador utilizados e pela alteração de como as variáveis eram declaradas de forma a adaptar o seu funcionamento à nova plataforma.

Os registos dos controladores CAN utilizam nomenclaturas diferentes no DSP, sendo por isso necessário adaptar o código de forma a realizar a transmissão e recepção de mensagens CAN de forma correcta. A sua verificação dividiu-se em duas fases, sendo a primeira dedicada ao envio de mensagens, verificando que valores estavam de facto a ser escritos nos registos CAN. Mais tarde, com o envio de mensagens já a funcionar, foi verificada a recepção através da impressão de valores na porta série, via função *main*.

Com os módulos de transmissão e recepção do CAN a funcionar, foi necessário alterar a rotina do *timer*, realocando trechos de código e alterando a nomenclatura dos

registos utilizados da mesma forma que para os registos CAN. A configuração do *timer* no arranque do nodo, também sofreu alterações especialmente no que diz respeito aos valores de *prescale*.

As rotinas de serviço à interrupção para o *timer* e controlador CAN utilizados foram ainda definidos como operações de prioridade máxima, com o objectivo de obter desempenho máximo na execução do protocolo.

De seguida foi necessário garantir o funcionamento correcto das rotinas de serviço à interrupção, configurando o sistema de forma correcta, garantindo que as interrupções eram activadas e desactivadas nos instantes correctos.

Nos restantes módulos, as alterações realizadas foram mínimas, existindo mesmo alguns que não necessitaram de quaisquer modificações, pois a sua programação baseava-se apenas na linguagem C, sem fazer uso de registos do processador.

As funções de *gateway* do *Slave* não foram implementadas, pois o objectivo principal deste trabalho estava focado na implementação e teste do subsistema de mensagens síncronas. Um *gateway* é definido como uma ponte entre sistemas diferentes e no exemplo da figura 3.4 fará de ponte entre um computador pessoal e um sistema CAN.

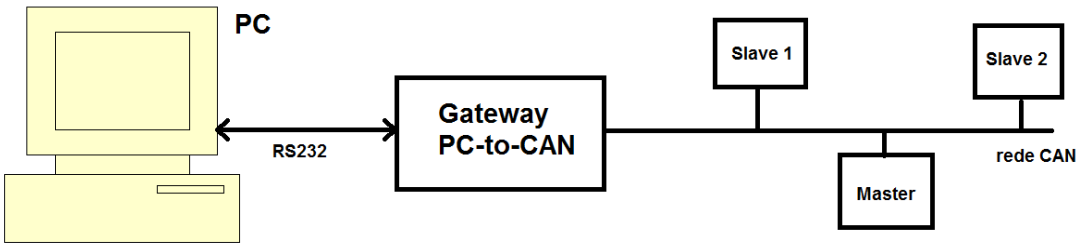

Figura 3.4 – Exemplo de um *Gateway* PC-to-CAN.

Uma vez que o trabalho aqui realizado incide exclusivamente na implementação dos nodos *Slave* do protocolo FTT-CAN em barramento replicado e uma vez que o nodo *Master* já se encontra implementado nesta plataforma, alguns dos módulos existentes estavam disponíveis e foram utilizados neste trabalho, como foi o caso dos módulos de configuração e escrita dos controladores CAN e porta série.

No final desta fase, o sistema implementado para DSP funcionava da mesma forma que um sistema FTT-CAN convencional, recorrendo apenas a um controlador CAN,

com a excepção das funcionalidades de *gateway*, embora nesta versão apenas esteja considerada uma velocidade de 250kbps para a transmissão em CAN.

3.3. Redundância e Largura de Banda em FTT-CAN

Um dos principais requisitos de um sistema de comunicação de tempo-real é a capacidade de tolerar a ocorrência de falhas. Como dito anteriormente, o protocolo FTT-CAN, na sua implementação actual, não possui mecanismos de redundância que permitam ao sistema continuar a funcionar correctamente em situações de falha do barramento, além disso as velocidades praticadas pelo sistema actual são bastante baixas devido ao *overhead* computacional introduzido nos processadores pelo *middleware* do FTT e pelo *overhead* introduzido na rede pelas TMs e mensagens de controlo [6]. De forma a melhorar o desempenho do FTT-CAN foi proposta uma implementação em barramento replicado.

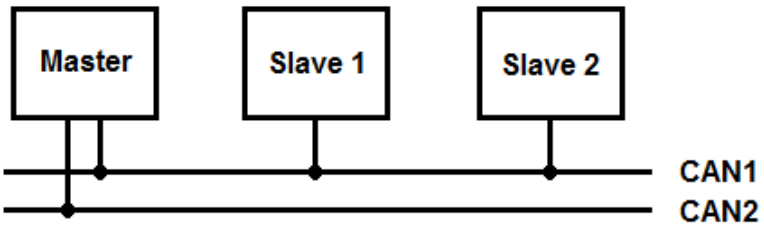

Figura 3.5 – FTT-CAN implementado em DSP utilizando barramento simples.

Nesta fase, o sistema em DSP opera como o FTT-CAN convencional, recorrendo apenas a um dos barramentos disponíveis como ilustrado na figura 3.5, com a excepção das funções de *gateway* como referido anteriormente. Sendo esta parte do trabalho dedicada a implementar os módulos de gestão do segundo barramento, que deverão funcionar de forma semelhante à do primeiro barramento. Isto é, realizar as tarefas de descodificação da TM, definição das janelas temporais do EC, programação e disparo de *timers* e envio das mensagens respectivas.

O *Master* implementado em dsPIC30F6012A envia TMs diferentes em cada barramento, permitindo-lhe gerir de forma dinâmica a atribuição de barramentos às mensagens. Deste modo o funcionamento dos *Slaves* será totalmente transparente, apenas necessitando de descodificar e seguir a TM, de forma a enviar as mensagens no barramento que o *Master* lhe indica.

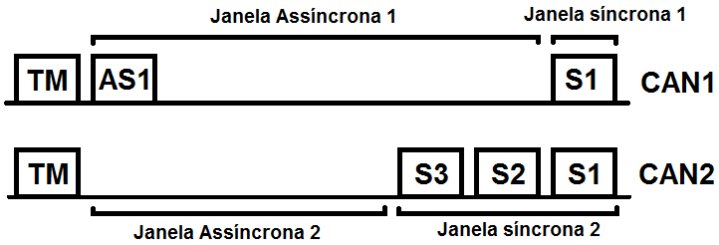

Figura 3.6 – Esquema temporal das janelas temporais em ambos os barramentos.

Na figura 3.6 é possível verificar como as janelas temporais podem ser diferentes em ambos os barramentos no mesmo instante. Ao receber cada TM, é necessário um mecanismo para efectuar a gestão das janelas de cada barramento, iniciando a transmissão das mensagens síncronas no instante correcto. Deste modo foi utilizado um *timer* por barramento, estes serão programados de acordo com a duração da janela síncrona codificada na TM recebida nesse barramento. Cada *timer* poderá disparar em instantes diferentes dando início à transmissão das mensagens síncronas respectivas. A opção pela utilização de dois *timers* foi devidamente ponderada e foi tomada pois apresenta uma simplicidade significativa face à alternativa de utilizar apenas um *timer* com sucessivas reprogramações, além disso o número de *timers* que permanecem disponíveis para as aplicações do utilizador é ainda superior à implementação em PIC18F258. Porém, esta situação pode ser vista como um possível ponto de optimização e objecto de uma análise mais dedicada no futuro.

Para além dos módulos de gestão das janelas temporais, o funcionamento do FTT-CAN utiliza outros módulos, como por exemplo a gestão de recepção e envio de mensagens CAN. O funcionamento destes módulos está descrito nos fluxogramas que se seguem.

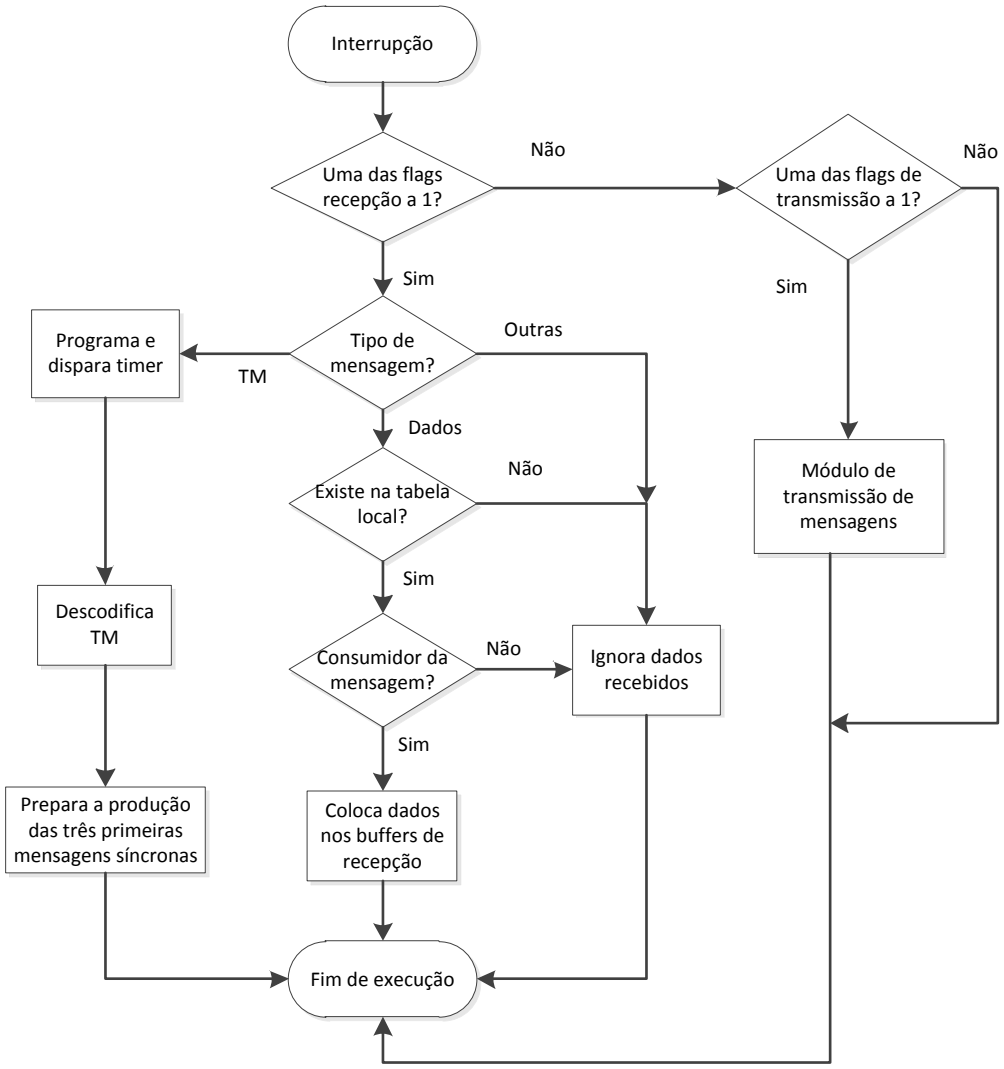

Figura 3.7 – Fluxograma da recepção de mensagens FTT-CAN.

O fluxograma da figura 3.7 descreve a sequência de operações desencadeada pela recepção de uma mensagem, seja ela TM, síncrona ou assíncrona, sendo que os módulos de ambos os barramentos funcionam desta forma.

Ao ser gerada uma interrupção o algoritmo verifica as *flags* de recepção e caso estejam activas lê a mensagem dos *buffers* CAN. De seguida é efectuada uma verificação do tipo de mensagem, caso seja TM procede com a sua descodificação, programação do *timer* e prepara o envio das três primeiras mensagens síncronas, preenchendo os três *buffers* de produção com o conteúdo das mensagens, para diminuir o *overhead* de processamento no instante de envio. No caso da mensagem recebida conter dados,

existir nas tabelas locais e o nodo ser consumidor da mesma, procede à sua salvaguarda nos *buffers* FTT respectivos, caso contrário ignora os dados recebidos.

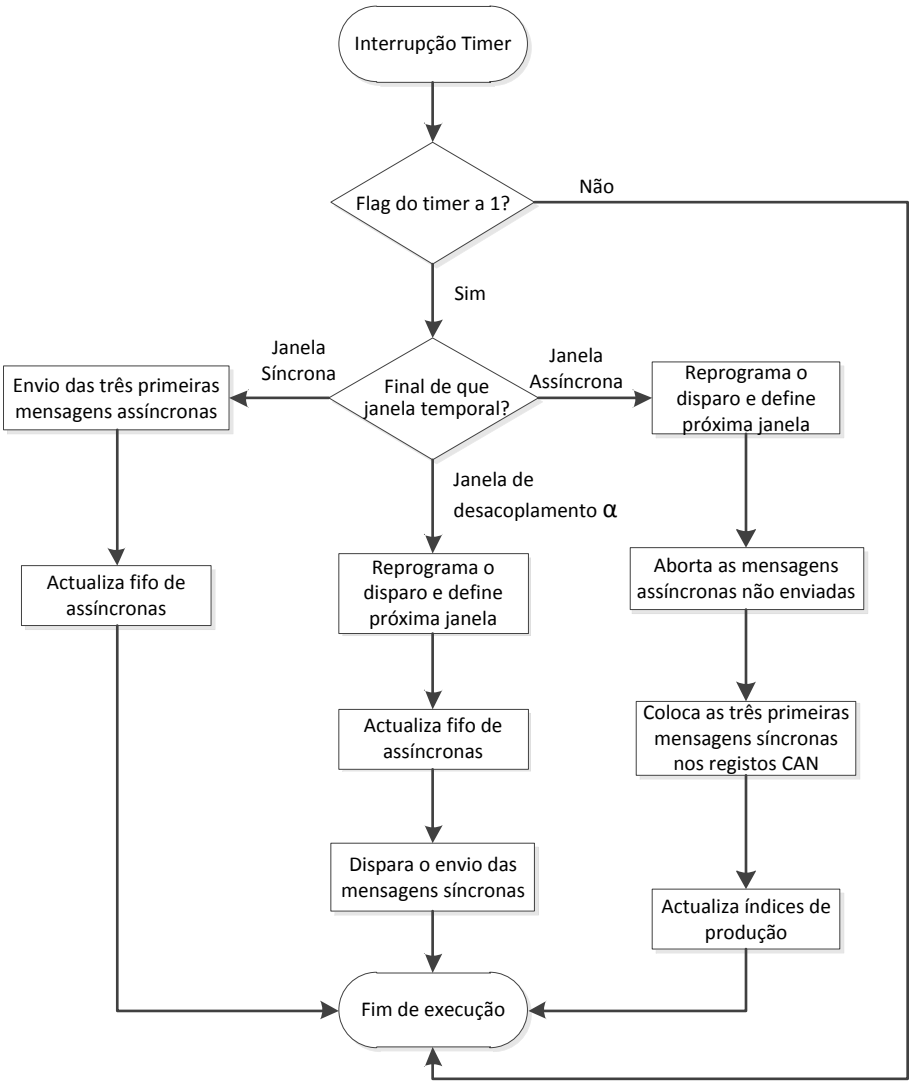

Figura 3.8 – Fluxograma do algoritmo de gestão de janelas temporais por *timer*.

A figura 3.8 exemplifica a forma como as rotinas dos *timers* gerem as janelas temporais e como se reprogramam de acordo com a dinâmica do sistema. Aqui são definidas três janelas em vez de duas, a janela de desacoplamento (α) tem como função separar a janela assíncrona da janela síncrona e tem a mesma duração que a mensagem CAN mais longa, de forma a evitar que existam mensagens assíncronas que possam ocupar qualquer tempo que esteja alojado para a janela síncrona. Por exemplo, se o envio

de uma mensagem assíncrona terminar no instante que o *timer* dispara para efectuar a transição de janelas e não existir esta janela intermédia, a mensagem vai acabar de ser transmitida já dentro da janela síncrona, prejudicando o agendamento previsto. É preciso ter em conta o instante do disparo do *timer*, uma vez que este se dá no final da janela que decorre, irá fazer o processamento correspondente à janela seguinte. A sequência de eventos está exemplificada na figura 3.9.

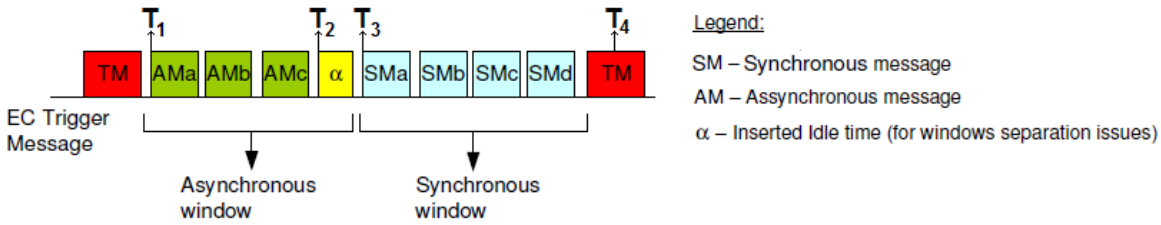

Figura 3.9 – Instantes de interesse na execução do *timer*.

No instante T_1 o *timer* será programado para definir a janela assíncrona, de acordo com o tamanho da janela síncrona codificado na TM, em que a duração da janela assíncrona é dado pelo tamanho do EC subtraído da duração da janela síncrona. Em T_2 o *timer* dispara de forma a controlar a transição entre janelas, aborta as mensagens que não irão caber na janela assíncrona, actualiza os respectivos *buffers* e prepara a produção das três primeiras mensagens síncronas. No instante T_3 é detectado o fim da janela de transição α e é neste momento que se dá início à transmissão de mensagens síncronas. Em T_4, após terminar a janela síncrona e ainda no decorrer do envio da TM, por parte do *Master*, é preparado o envio das três primeiras mensagens assíncronas que serão enviadas assim que começar o próximo EC.

As rotinas dos módulos CAN possuem ainda a responsabilidade de gerir a transmissão de mensagens. Quando uma mensagem fica pronta a enviar é gerada uma interrupção num dos módulos CAN, o algoritmo irá avaliar qual a janela temporal actual e enviar as mensagens de acordo com o resultado. Este funcionamento pode ser observado no fluxograma da figura 3.10.

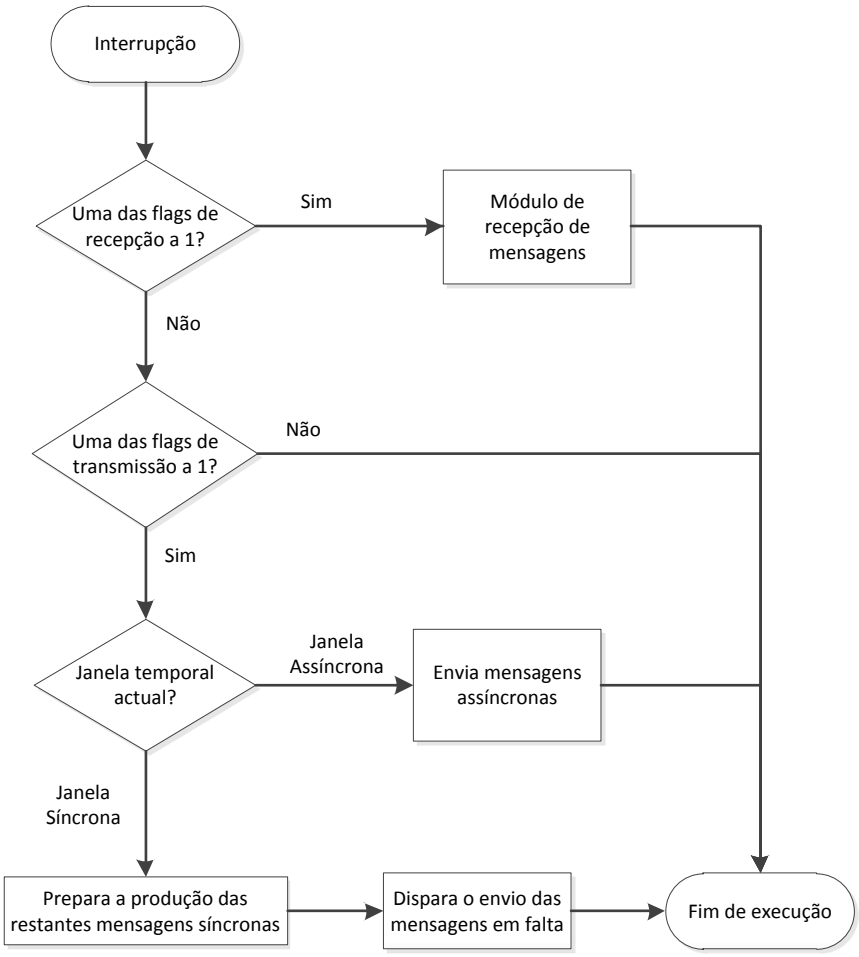

Figura 3.10 – Transmissão de mensagens FTT-CAN.

Para que o envio de mensagens possa ser realizado, é necessário adicionar as mensagens a uma das tabelas locais. Cada tabela destina-se a gerir um tipo de mensagens, uma irá conter todas as mensagens síncronas e outra as assíncronas para utilização do nodo, quer para consumo quer para produção. Ao adicionar as mensagens numa tabela é necessário especificar se o nodo é consumidor ou produtor da mensagem e qual o tamanho da mensagem. No que diz respeito a mensagens síncronas, o envio será sempre feito por onde a TM definir. Deste modo as funções de adição de variáveis nas tabelas não necessitaram de alterações.

Capítulo 4

Resultados

4.1. Funcionamento

4.1.1. Verificação do envio de mensagens e descodificação da *Trigger Message*

De forma a verificar o envio e recepção de mensagens síncronas, foi realizado um ensaio utilizando uma montagem como exemplificado no esquema da figura 4.1.

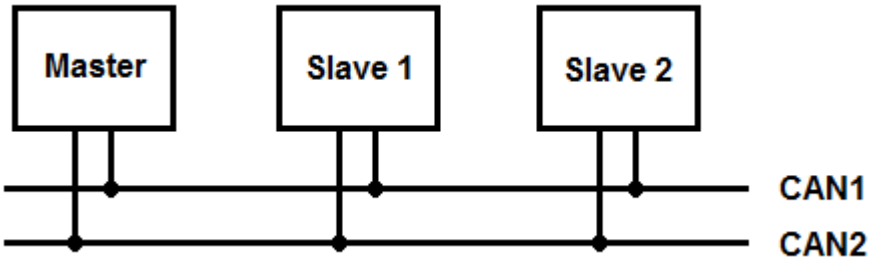

Figura 4.1 – Esquema da montagem utilizada.

A taxa de transmissão utilizada nos ensaios foi 250kbps. Foram então adicionados três tipos de variáveis síncronas na tabela SRT do nodo *Master* possuindo as características descritas na Tabela 4.1.

ID - Tipo de Mensagem	Período	Tamanho	No(s) Barramento(s):
0 – Mensagem Crítica	1	8 *Bytes*	CAN1 & CAN2
1 – Não crítica	1	8 *Bytes*	CAN1
2 – Não crítica	1	8 *Bytes*	CAN2

Tabela 4.1 – Caracterização das variáveis adicionadas na SRT para teste.

A mensagem crítica deverá ser transmitida pelos dois barramentos e as mensagens não críticas deverão seguir uma por barramento. Uma vez que os períodos definidos são 1, todas as mensagens deverão aparecer a cada EC. Definiu-se então um nodo como produtor das três mensagens e outro como consumidor também das três

mensagens e um EC de 5ms. Com este procedimento deveriam ser observadas duas mensagens por barramento. De forma a verificar a existência das mensagens no barramento, utilizou-se um osciloscópio e a impressão de valores no ecrã.

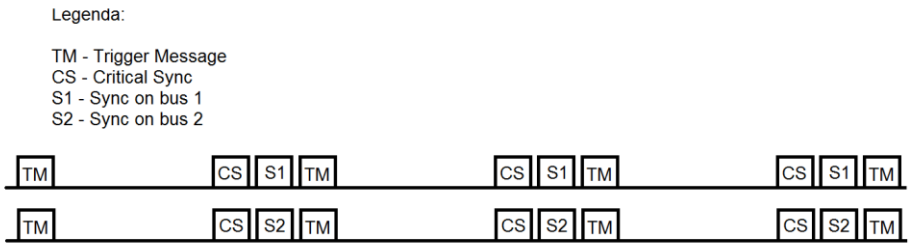

Figura 4.2 – Mensagens visualizadas no barramento através do osciloscópio.

Analisando a figura 4.2, que identifica as mensagens presentes no barramento, verifica-se que de facto existem duas mensagens de dados por EC. Pelo que é possível concluir que o sistema tem um comportamento semelhante ao que seria de esperar uma vez que apresenta o mesmo número de mensagens que a configuração escolhida para cada barramento.

De forma a despistar possíveis falhas na descodificação da TM, foi repetido o processo mas agora utilizando valores diferentes para o período de cada mensagem. Na tabela 3 estão registados os novos valores utilizados na realização deste teste.

ID - Tipo de Mensagem	Período	Tamanho	No(s) Barramento(s):
0 – Mensagem Crítica	1	8 *Bytes*	CAN1 & CAN2
1 – Não crítica	2	8 *Bytes*	CAN1
2 – Não crítica	3	8 *Bytes*	CAN2

Tabela 4.2 – Nova caracterização das variáveis adicionadas na SRT para teste.

Nesta situação deveria ser obtido um comportamento completamente diferente, uma vez que os períodos foram alterados. Cada mensagem crítica deve continuar a ser enviada a cada EC, mas as não críticas apenas devem ser enviadas a cada 2 ECs e 3 ECs, nos barramentos CAN1 e CAN2 respectivamente. A verificação das mensagens recebidas

é feita novamente pelas impressões do nodo consumidor e pela visualização das mesmas nos barramentos. Sendo os respectivos resultados apresentados na figura 4.3.

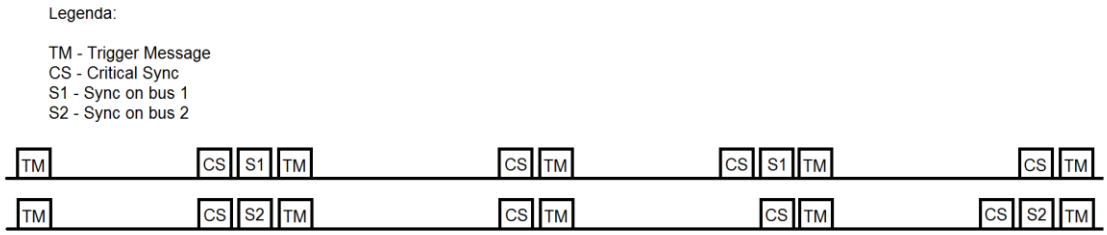

Figura 4.3 – Mensagens visualizadas no barramento através do osciloscópio.

Comparando os resultados obtidos em ambos os testes anteriores, verifica-se que as mensagens seguem o padrão imposto pelos respectivos períodos definidos no nodo *Master* como seria de esperar. De onde se pode concluir que tanto a descodificação da TM como o envio das mensagens se está a fazer de forma aceitável.

4.1.2. Verificação da correcção de escrita e leitura nos *buffers* CAN.

Uma vez que o envio das mensagens foi verificado, é agora necessário verificar se o conteúdo das mensagens a enviar e a receber correspondem de facto ao que é definido. Utilizando o mesmo esquema e taxa de transmissão que no ponto anterior, foi criada uma aplicação que realiza o envio de duas mensagens não críticas, enviando uma em cada canal. Para esta experiência foi definido um EC com duração de 5 ms, mensagens com um tamanho de 8 *bytes* e um período de 500 ms.

Um período desta ordem será útil para verificar se existem falhas na passagem de valores entre *buffers* e se estão a ocorrer nos instantes definidos, pois deste modo a velocidade de operação do FTT será muito superior ao período da mensagem e no caso de haver problemas de escrita nos *buffers* irão tornar-se óbvios rapidamente. Na figura 4.4 está ilustrado o *timeline* de eventos que ocorrem durante dois segundos neste procedimento.

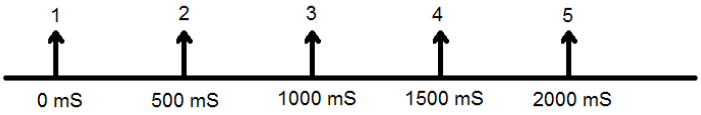

Figura 4.4 – *Timeline* de eventos.

A cada uma das mensagens foi atribuído um contador de 8 *bits*, sendo um inicializado com 0 e o outro com 100. A cada segundo (instantes 3 e 5 da figura 4.4) os contadores são incrementados e a mensagem preenchida com {X,X+1,…,X+7}. A cada 500ms (instantes 2,3,4 e 5 da figura 4.4) é enviada uma mensagem com o conteúdo definido nesse instante. Assim que os dados são recebidos, o nodo consumidor envia os dados recebidos pela porta série. O resultado obtido está ilustrado na imagem seguinte.

```
Sync1 -> '|B7: 8 |B6: 7 |B5: 6 |B4: 5 |B3: 4 |B2: 3 |B1: 2 |B0: 1 |'.
Sync2 -> '|B7: 108 |B6: 107 |B5: 106 |B4: 105 |B3: 104 |B2: 103 |B1: 102 |B0: 101 |'.
Sync1 -> '|B7: 8 |B6: 7 |B5: 6 |B4: 5 |B3: 4 |B2: 3 |B1: 2 |B0: 1 |'.
Sync2 -> '|B7: 108 |B6: 107 |B5: 106 |B4: 105 |B3: 104 |B2: 103 |B1: 102 |B0: 101 |'.
Sync1 -> '|B7: 9 |B6: 8 |B5: 7 |B4: 6 |B3: 5 |B2: 4 |B1: 3 |B0: 2 |'.
Sync2 -> '|B7: 109 |B6: 108 |B5: 107 |B4: 106 |B3: 105 |B2: 104 |B1: 103 |B0: 102 |'.
Sync1 -> '|B7: 9 |B6: 8 |B5: 7 |B4: 6 |B3: 5 |B2: 4 |B1: 3 |B0: 2 |'.
Sync2 -> '|B7: 109 |B6: 108 |B5: 107 |B4: 106 |B3: 105 |B2: 104 |B1: 103 |B0: 102 |'.
Sync1 -> '|B7: 10 |B6: 9 |B5: 8 |B4: 7 |B3: 6 |B2: 5 |B1: 4 |B0: 3 |'.
Sync2 -> '|B7: 110 |B6: 109 |B5: 108 |B4: 107 |B3: 106 |B2: 105 |B1: 104 |B0: 103 |'.
Sync1 -> '|B7: 10 |B6: 9 |B5: 8 |B4: 7 |B3: 6 |B2: 5 |B1: 4 |B0: 3 |'.
Sync2 -> '|B7: 110 |B6: 109 |B5: 108 |B4: 107 |B3: 106 |B2: 105 |B1: 104 |B0: 103 |'.
Sync1 -> '|B7: 11 |B6: 10 |B5: 9 |B4: 8 |B3: 7 |B2: 6 |B1: 5 |B0: 4 |'.
Sync2 -> '|B7: 111 |B6: 110 |B5: 109 |B4: 108 |B3: 107 |B2: 106 |B1: 105 |B0: 104 |'.
Sync1 -> '|B7: 11 |B6: 10 |B5: 9 |B4: 8 |B3: 7 |B2: 6 |B1: 5 |B0: 4 |'.
Sync2 -> '|B7: 111 |B6: 110 |B5: 109 |B4: 108 |B3: 107 |B2: 106 |B1: 105 |B0: 104 |'.
Sync1 -> '|B7: 12 |B6: 11 |B5: 10 |B4: 9 |B3: 8 |B2: 7 |B1: 6 |B0: 5 |'.
Sync2 -> '|B7: 112 |B6: 111 |B5: 110 |B4: 109 |B3: 108 |B2: 107 |B1: 106 |B0: 105 |'.
Sync1 -> '|B7: 12 |B6: 11 |B5: 10 |B4: 9 |B3: 8 |B2: 7 |B1: 6 |B0: 5 |'.
Sync2 -> '|B7: 112 |B6: 111 |B5: 110 |B4: 109 |B3: 108 |B2: 107 |B1: 106 |B0: 105 |'.
```

Figura 4.5 – Resultados obtidos na verificação de correcção de escrita/leitura nos *buffers* CAN.

Na figura 4.5, o conteúdo de cada Byte que constitui a mensagem está identificado por Bx, onde x identifica o Byte a que se refere. Na primeira linha temos portanto uma mensagem cujo conteúdo é no primeiro byte “1”, no segundo “2”, no terceiro “3” e assim sucessivamente, sendo esse preenchimento ilustrado na figura 4.6.

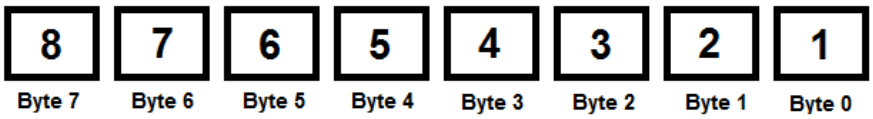

Figura 4.6 – Conteúdo da primeira mensagem recebida no CAN1.

São sempre recebidas duas mensagens com o mesmo conteúdo a cada segundo, uma vez que o conteúdo da mensagem é actualizado apenas a cada dois envios. Da análise da figura verifica-se a coerência entre esta afirmação e os dados obtidos. Assim é possível concluir que a escrita e leitura nos *buffers* CAN está a ser realizada de forma correcta, visto que todos os *bytes* correspondem ao que era esperado, em ambos os controladores.

4.1.3. Verificação do mecanismo de redundância

De forma a pôr à prova a capacidade do sistema tolerar falhas do barramento foram realizados alguns testes que consistiram em simular a ocorrência de falha em determinado barramento removendo por completo o barramento a testar, verificando e registando o comportamento do sistema. A tabela seguinte regista os tipos de mensagens síncronas que foram utilizados neste procedimento:

ID - Tipo de Mensagem	Período	No(s) Barramento(s)
0 – Mensagem Crítica	1	CAN 1 & CAN 2
1 – Não crítica	1	CAN 1
2 – Não crítica	1	CAN 2

Tabela 4.3 – Atribuição de barramentos a três tipos de mensagens síncronas.

A tabela 4.4 contém o registo dos comportamentos verificados no decorrer da experiência, enviando as três mensagens em cada EC.

	Mensagens recebidas	
Acção	CAN 1	CAN 2
Ambos Ligados	**Crítica** + Não Crítica 1	**Crítica** + Não Crítica 2
CAN 1 Desligado	-	**Crítica** + Não Crítica 2
CAN 2 Desligado	**Crítica** + Não Crítica 1	-

Tabela 4.4 – Registo das mensagens recebidas quando se desliga determinado barramento.

Da análise da tabela 4.4 é possível apurar que a mensagem crítica é aquela que atinge o nodo consumidor em qualquer um dos três cenários, mesmo aqueles que

simulam a falha de um dos barramentos. Assim é garantindo que este tipo de mensagem é tratado pelo sistema com uma importância acrescida. O que se verificou dos dados obtidos é que as mensagens críticas chegam sempre ao destino e as mensagens não críticas apenas atingem o destino se o barramento por onde são enviadas não apresentar problemas.

Na verdade este comportamento não era esperado, uma vez que o *Master* deveria mudar as mensagens de barramento quando verifica a falha do mesmo, o que não sucedeu. Ao se efectuar uma verificação rápida ao código fonte do *Master* tornou-se óbvia a razão deste comportamento, o trecho de código responsável pela mudança de barramento encontrava-se comentado daí não ocorrer mudança de barramento. Ao incluir esse código na compilação do *Master* as mensagens que correspondiam ao barramento em falha eram transportadas para o barramento funcional.

4.1.4. Medida dos tempos de processamento nos *Slaves*

Com a transmissão e recepção de mensagens síncronas a ser realizada de forma correcta é agora necessário avaliar o comportamento temporal do sistema. Para isso foi preparado um procedimento com o objectivo de medir a que instantes ocorrem determinados eventos do sistema. O mecanismo de medição foi implementado recorrendo a um *timer* configurado para operar com resolução máxima, bastando para isso utilizar um *prescale* de 1:1. Uma vez que o DSP está configurado para operar a 80MHz, a frequência de operação do *timer* será:

$$frequência\ do\ timer = \frac{80\ MHz}{4\ x\ prescale\ (1:1)} = 20\ MHz$$

$$resolução\ do\ timer = \frac{1}{20\ MHz} = 50\ nS$$

O *timer* de 16 *bits* foi configurado para disparar apenas depois de fazer uma contagem completa, isto é 65536. Uma vez que o EC tem duração de 5 ms e a resolução escolhida é 50 ns, os 16 *bits* não são suficientes para realizar a contagem durante todo o EC, deste modo foi utilizada uma variável na rotina de atendimento do *timer* para

implementar o *bit* em falta. Para não existir inconsistência dos valores medidos, esta rotina foi definida com prioridade superior às restantes.

O *timer* deverá ser activado no instante em que é recebida uma TM e nos instantes a medir, o valor do *timer* é passado para um *buffer* intermédio e posteriormente realizado o envio desses valores pela porta série através da função *main*. No computador, foi desenvolvido um programa simples que comunica com a porta série e recolhe o número de amostras definido pelo utilizador guardando o resultado num ficheiro de texto. O desenvolvimento deste programa assenta na utilização do módulo *SerialPort* já pronto a utilizar e implementado pela gigante do *software*, Microsoft. De um modo muito simples, este módulo permite ligar à porta série, ler e escrever caracteres.

A taxa de transmissão utilizada para o barramento CAN ao longo das medições foi de 250kbps e o tamanho das mensagens de 8 *Bytes*. Para efeitos de análise, foram utilizadas cinco mil amostras de forma a obter qualidade na análise realizada. Os dados foram depois tratados em Matlab de forma a obter a média, desvio padrão, máximo e mínimo dos valores a medir.

Os valores que interessam medir estão ligados à descodificação da TM e ao envio de mensagens síncronas. Relativamente à TM, convém apurar a latência introduzida pela sua descodificação, portanto assim que a descodificação estiver concluída, o valor do *timer* deverá ser registado. Para verificar se as mensagens síncronas estão a ser enviadas no instante correcto e de forma a quantificar o *jitter* introduzido no instante de envio, o valor do *timer* deve ser lido assim que o pedido de transmissão da primeira mensagem síncrona esteja concluído. O diagrama da figura 4.7 enquadra as medições feitas no decorrer do EC, sendo que a duração das medições estão assinaladas por 1 e 2.

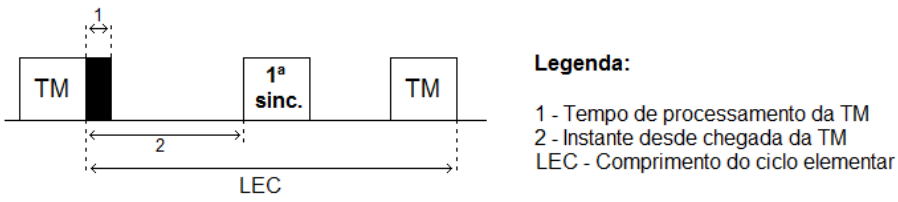

Figura 4.7 – Diagrama temporal indicando instantes de medição.

Para oferecer uma caracterização mais completa do funcionamento do sistema, este teste será ainda estendido a vários cenários, onde o *Master* escalonará diferentes

conjuntos de mensagens e o *Slave* fará também o envio de diversos conjuntos de mensagens. Para cada cenário foi construído um histograma das amostras recolhidas e registado numa tabela os respectivos valores de média, desvio padrão, máximo e mínimo. Este teste foi realizado para ambos os barramentos e para a versão PIC18, os histogramas são apresentados com o número de ocorrências em escala logarítmica para facilitar a visualização dos valores.

Cenário a) – *Slave* produz uma mensagem síncrona e *Master* escalona duas mensagens – barramento 1.

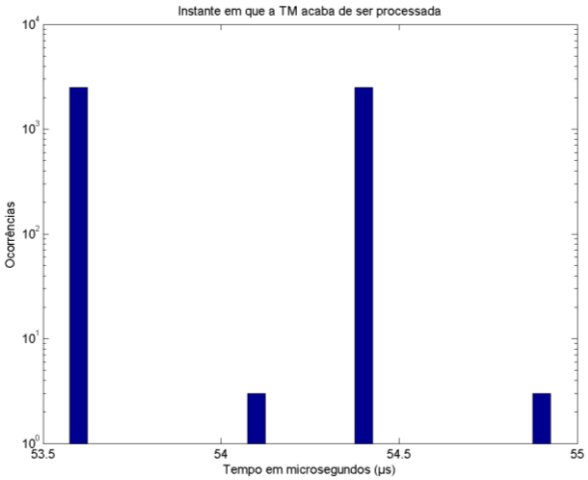

Figura 4.8 – Histograma dos tempos de processamento da TM para o cenário a) – barramento 1.

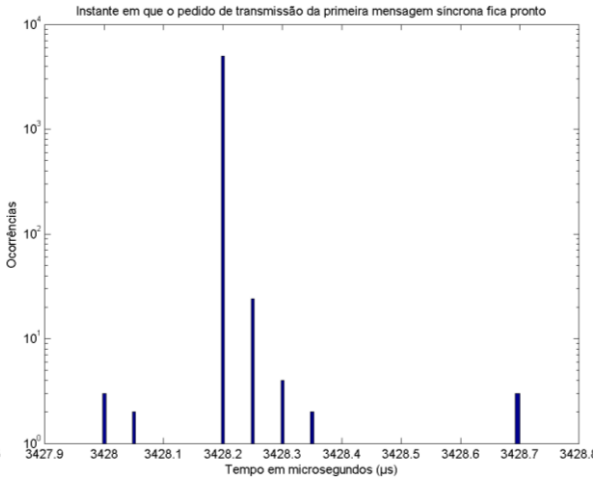

Figura 4.9 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário a) – barramento 1.

No cenário a), o *Slave* produz uma mensagem síncrona e o *Master* escalona duas mensagens. A figura 4.8 mostra que os resultados obtidos são bastante próximos e no caso de produção de uma mensagem a latência introduzida pela descodificação da TM representa aproximadamente 54 µs. Quanto à figura 4.9, o *jitter* que afecta o instante em que o pedido da primeira mensagem síncrona fica pronto, é também muito baixo. Quanto ao instante a que ocorre verifica-se que possui um valor próximo do esperado uma vez que o *timer* começa a contagem quando a TM é completamente recebida, isto é, cerca de 500 µs após o início do EC. Deste modo o EC terminará quando o *timer* atingir uma contagem de cerca de 4,5 ms. Como cada mensagem síncrona tem também uma duração de cerca de 500 µs verifica-se que ao escalonar duas mensagens síncronas deverá ser registado um valor próximo de 3,5 ms para a contagem do *timer*.

Cenário b) – *Slave* produz uma mensagem síncrona e *Master* escalona quatro mensagens – barramento 1.

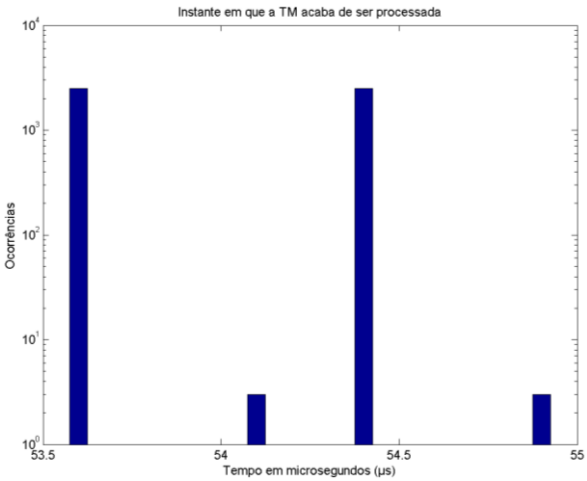

Figura 4.10 - Histograma dos tempos de processamento da TM para o cenário b) – barramento 1.

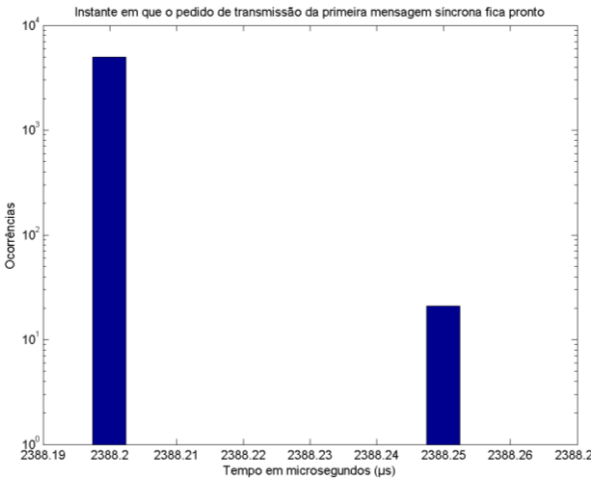

Figura 4.11 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário b) – barramento 1.

No cenário b), os valores da latência devido ao processamento da TM permanecem praticamente inalterados, uma vez que o *Slave* procede ao envio do mesmo número de mensagens síncronas que no caso anterior. O instante em que o início da primeira mensagem síncrona fica pronto é diminuído pela proporção correspondente ao tamanho de duas mensagens síncronas, as duas adicionadas ao escalonamento, o que vai de encontro ao que se esperava.

Cenário c) – *Slave* produz uma mensagem síncrona e *Master* escalona seis mensagens – barramento 1.

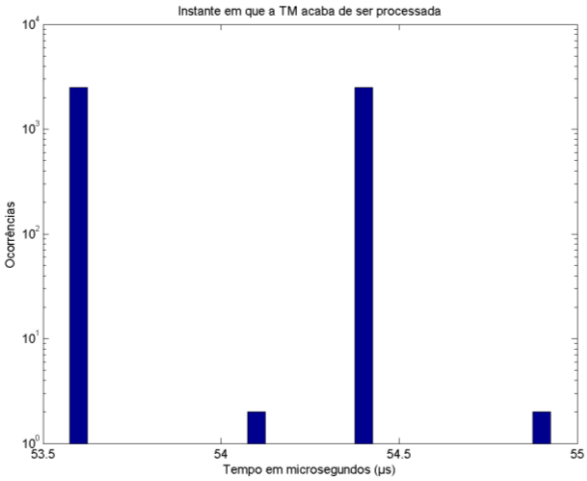

Figura 4.12 - Histograma de tempos de processamento da TM para o cenário c) – barramento 1.

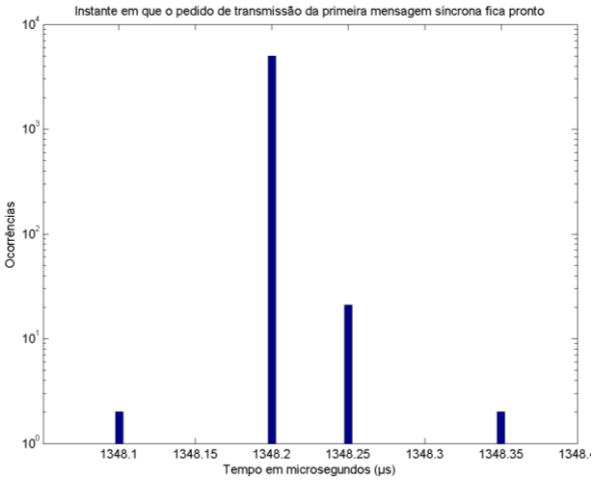

Figura 4.13 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário c) – barramento 1.

No cenário c), a latência permanece inalterada pelas mesmas razões que no cenário b), como se pode verificar pela figura 4.12. O instante em que a primeira mensagem síncrona fica pronto é novamente reduzido do tamanho das duas mensagens adicionadas ao escalonamento.

Cenário d) – *Slave* produz uma mensagem síncrona e *Master* escalona oito mensagens – barramento 1.

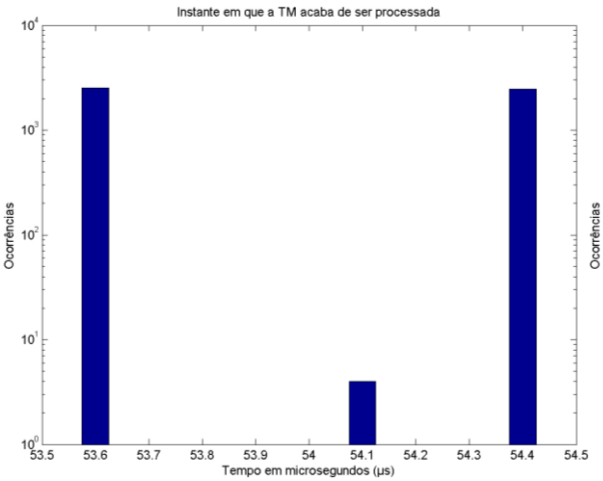

Figura 4.14 - Histograma de tempos de processamento da TM para o cenário d) – barramento 1.

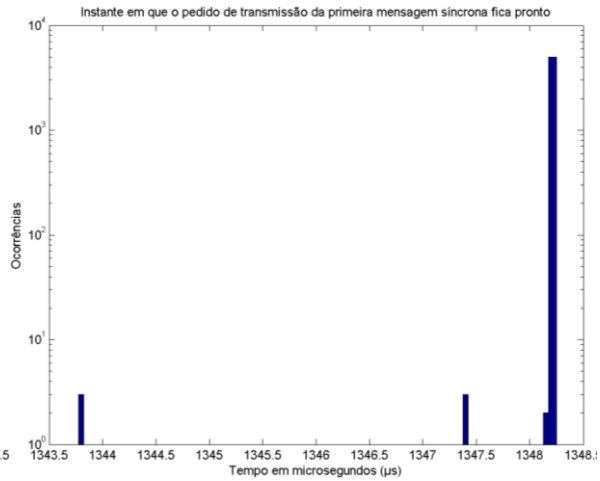

Figura 4.15 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário d) – barramento 1.

No cenário d), a alteração mais significativa está novamente relacionada com o instante em que o pedido da primeira mensagem síncrona fica pronto. Desta vez o que acontece é que as mensagens adicionadas ao escalonamento falham a *deadline* e o facto de terem sido adicionadas ao sistema não se reflecte no instante em que a janela síncrona se inicia. Os 1,35 ms que faltam preencher chegariam para colocar as duas mensagens, mas o FTT-CAN utiliza cerca de 500 µs para desacoplamento das janelas síncrona e assíncrona, ficando o restante para utilização da janela assíncrona.

Cenário e) – *Slave* produz duas mensagens síncronas e *Master* escalona duas mensagens – barramento 1.

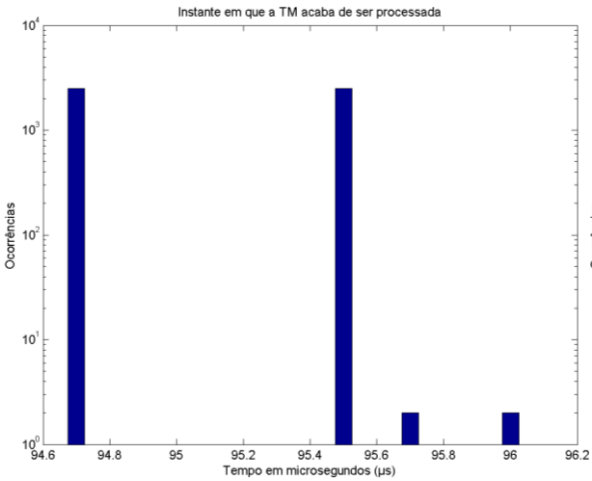

Figura 4.16 - Histograma de tempos de processamento da TM para o cenário e) – barramento 1.

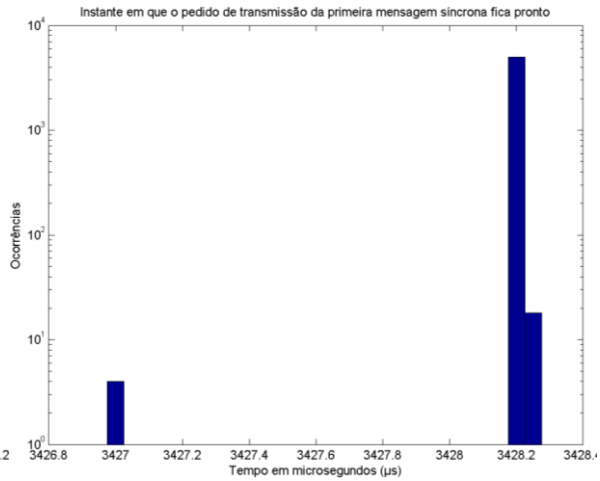

Figura 4.17 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário e) – barramento 1.

A diferença entre o cenário e) e o cenário a), é que neste caso o *Slave* produz duas mensagens em vez de uma, o que se reflecte na latência introduzida pelo processamento da TM. Agora a TM demora cerca de 95 µs em vez dos 54 µs do cenário a).

Cenário f) – *Slave* produz duas mensagens síncronas e *Master* escalona quatro mensagens – barramento 1.

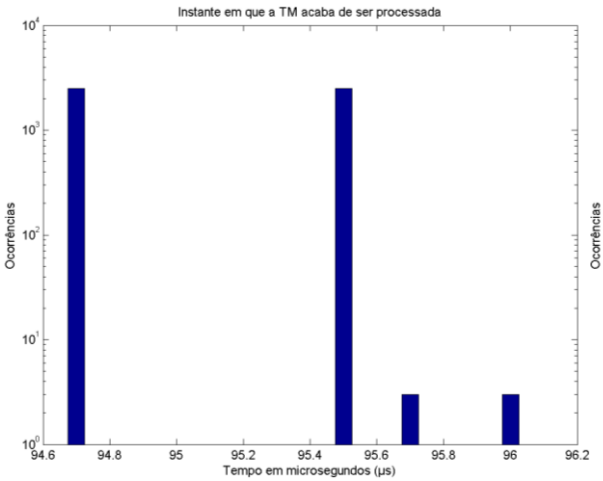

Figura 4.18 - Histograma de tempos de processamento da TM para o cenário f) – barramento 1.

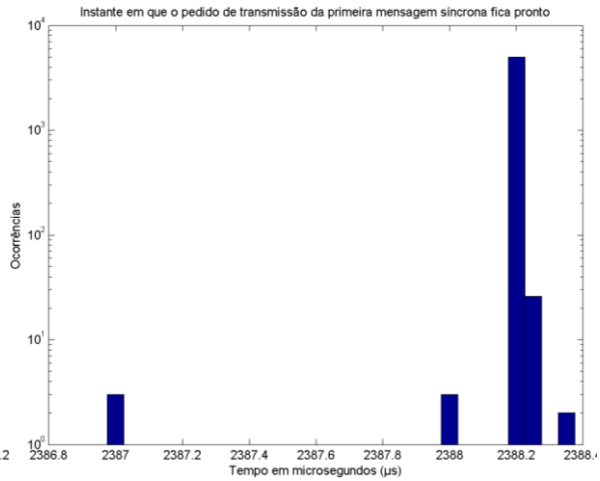

Figura 4.19 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário f) – barramento 1.

De modo semelhante ao que sucedeu no cenário e), no cenário f) a latência devido ao processamento da TM aumentou para cerca de 95 µs. O instante de início de transmissão da primeira mensagem síncrona também se altera de forma semelhante ao que ocorreu no cenário b) e pelas mesmas razões.

Cenário g) – *Slave* produz duas mensagens síncronas e *Master* escalona seis mensagens – barramento 1.

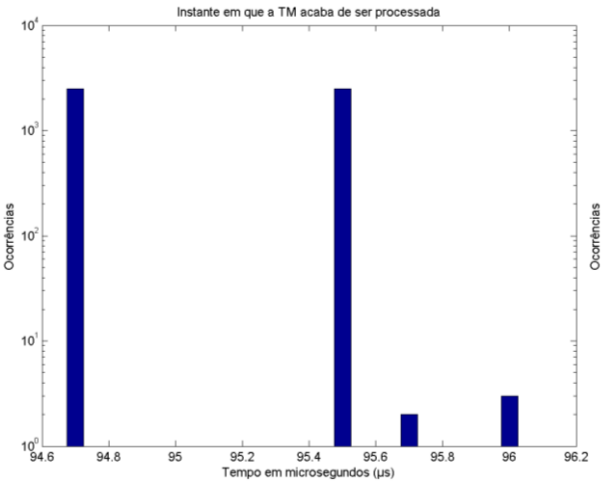

Figura 4.20 - Histograma de tempos de processamento da TM para o cenário g) – barramento 1.

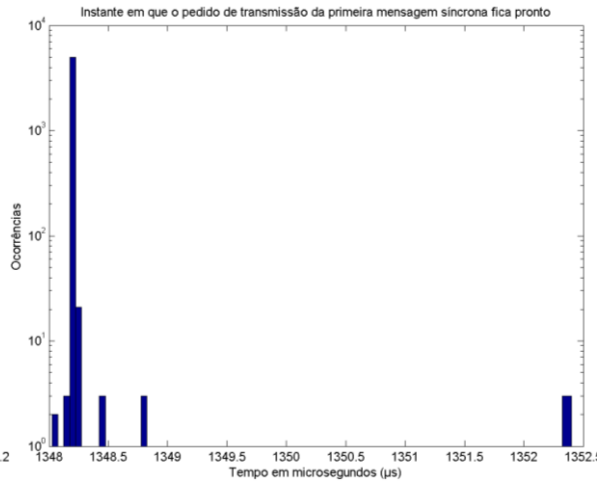

Figura 4.21 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário g) – barramento 1.

A latência introduzida no cenário g) é em tudo semelhante à introduzida no cenário e). O instante em que o pedido de transmissão da primeira mensagem síncrona do cenário g) é semelhante à situação do cenário c).

Cenário h) – *Slave* produz duas mensagens síncronas e *Master* escalona oito mensagens – barramento 1.

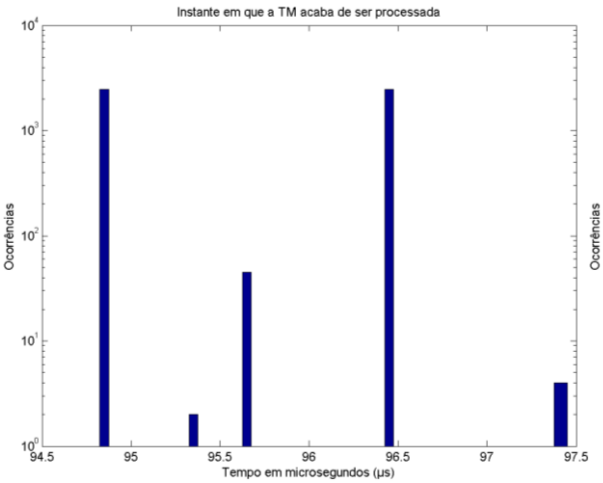

Figura 4.22 - Histograma de tempos de processamento da TM para o cenário h) – barramento 1.

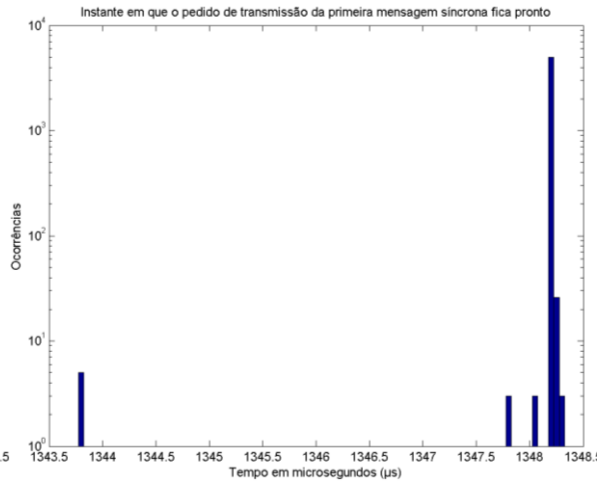

Figura 4.23 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário h) – barramento 1.

O tempo de processamento da TM no cenário h) permanece semelhante ao dos cenários anteriores, uma vez que o *Slave* produz o mesmo número de mensagens. O

instante em que o pedido de transmissão da primeira mensagem síncrona fica pronto é semelhante ao obtido para o cenário d), uma vez que ocorre a mesma situação de perda de *deadlines* devido ao número excessivo de mensagens a escalonar.

Cenário i) – *Slave* produz três mensagens síncronas e *Master* escalona quatro mensagens – barramento 1.

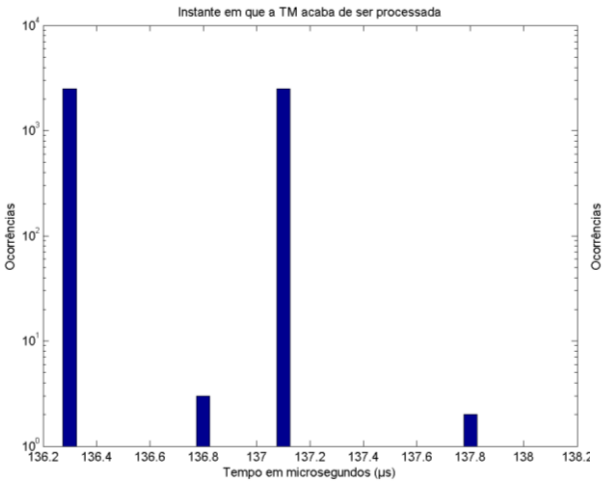

Figura 4.24 - Histograma de tempos de processamento da TM para o cenário i) – barramento 1.

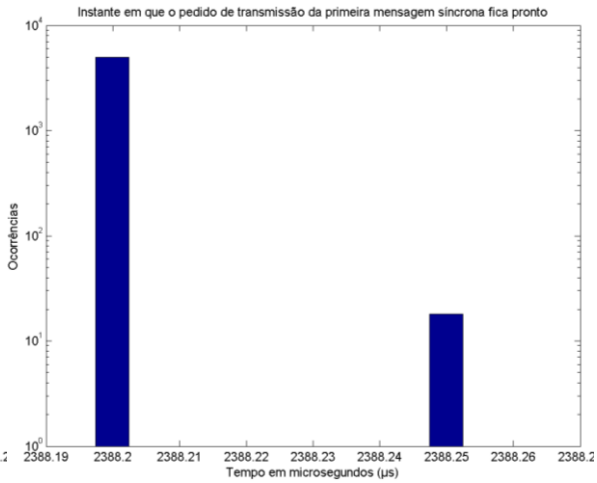

Figura 4.25 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário i) – barramento 1.

No cenário i) o que acontece é que a latência introduzida pelo processamento da TM aumenta devido ao aumento do número de mensagens a produzir pelo nodo. Quanto ao instante em que o pedido de transmissão da primeira mensagem síncrona, este apresenta valores semelhantes ao do cenário b), uma vez que o *Master* escalona o mesmo número de mensagens.

A razão para que não exista um ensaio com um *Slave* a produzir três mensagens enquanto o *Master* escalona apenas duas é porque esse cenário não faria sentido.

Cenário j) – *Slave* produz três mensagens síncronas e *Master* escalona seis mensagens – barramento 1.

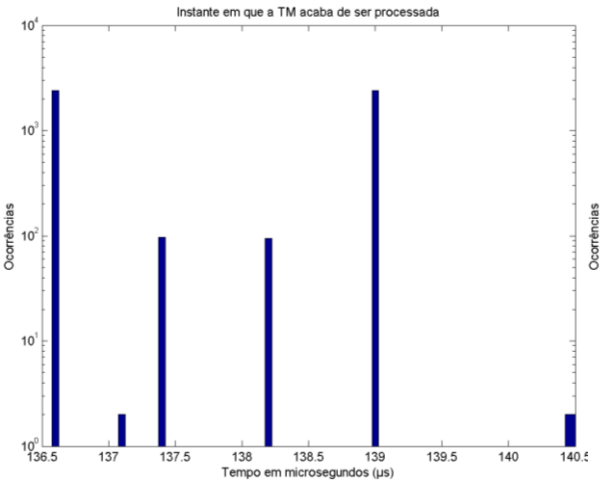

Figura 4.26 - Histograma de tempos de processamento da TM para o cenário j) – barramento 1.

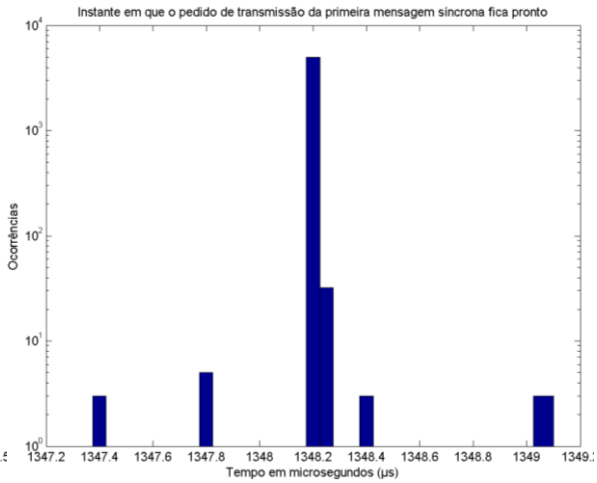

Figura 4.27 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário j) – barramento 1.

O cenário j) assemelha-se ao cenário i) em termos de latência devido ao processamento da TM. Quanto ao instante de início da janela síncrona apresenta valores mais relacionados com o cenário c).

Cenário k) – *Slave* produz três mensagens síncronas e *Master* escalona oito mensagens – barramento 1.

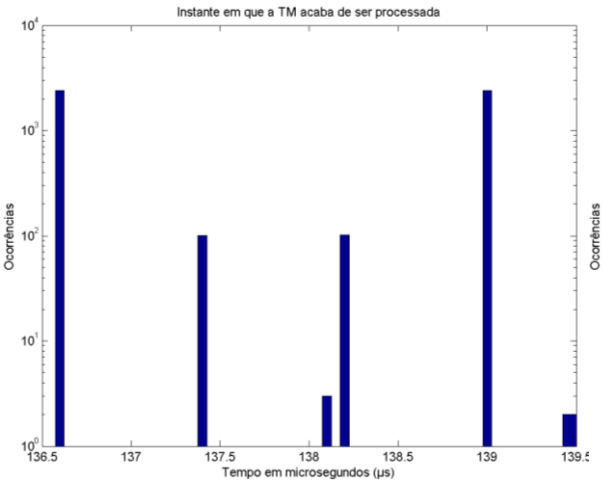

Figura 4.28 - Histograma de tempos de processamento da TM para o cenário k) – barramento 1.

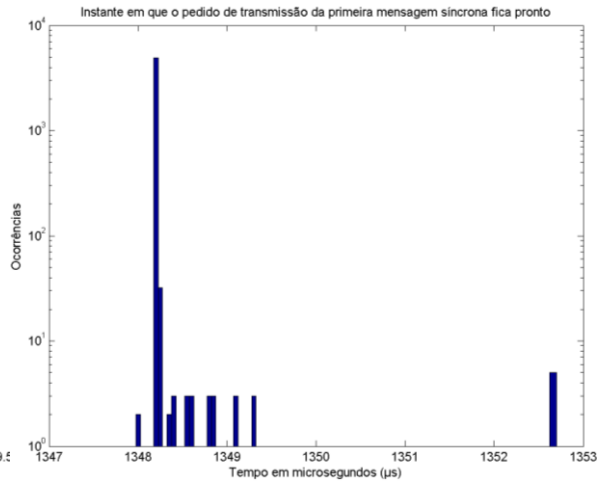

Figura 4.29 - Histograma de tempos para o pedido de transmissão da primeira mensagem síncrona no cenário k) – barramento 1.

O cenário k) apresenta uma latência correspondente ao processamento de três mensagens. Em termos de início da janela síncrona apresenta uma situação semelhante à descrita em d).

As tabelas 4.5 e 4.6 registam as caracterizações estatísticas mais relevantes para cada um dos cenários apresentados anteriormente.

Cenário	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Média	54,0	54,0	54,0	54,0	95,1	95,1	95,1	95,7	136,7	137,8	137,8
Desvio Padrão	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,8	0,4	1,2	1,2
Máximo	54,9	54,9	54,9	54,4	96,0	96,0	96,0	97,5	137,8	140,5	139,5
Mínimo	53,6	53,6	53,6	53,6	94,7	94,7	94,7	94,9	136,3	136,6	136,6

Tabela 4.5 – Dados estatísticos de cada cenário para tempos de processamento da TM do barramento 1. (valores apresentados em µs)

A tabela 4.5 identifica como valores aproximados dos tempos de processamento da TM 54 µs, 95 µs e 137 µs, para a produção de 1, 2 e 3 mensagens, respectivamente. A diferença entre estes valores representa o acréscimo de *overhead* introduzido por cada mensagem produzida, cerca de 40 µs.

Cenário	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Média	3428,2	2388,2	1348,2	1348,2	3428,2	2388,2	1348,2	1348,2	2388,2	1348,2	1348,2
Desvio Padrão	0,012	0,003	0,004	0,090	0,030	0,025	0,085	0,125	0,003	0,027	0,132
Máximo	3428,7	2388,3	1348,3	1348,3	3428,3	2388,4	1352,4	1348,3	2388,3	1349,1	1352,7
Mínimo	3428,0	2388,2	1348,1	1343,8	3427,0	2387,0	1348,1	1343,8	2388,2	1347,4	1348,0

Tabela 4.6 – Dados estatísticos de cada cenário para o instante em que o pedido da primeira mensagem síncrona fica pronto no barramento 1. (valores apresentados em µs)

Os dados obtidos apresentam uma grande proximidade nos seus valores. Inclusive, para situações como nos cenários a), b), c) e d) da tabela 4.5, por exemplo, em que o *Slave* produz o mesmo número de mensagens mas o *Master* escalona diferentes quantidades de mensagens. De forma semelhante, o instante em que o pedido de transmissão da primeira mensagem síncrona fica pronto, nos cenários b), f) e i) da tabela 4.6 por exemplo, os instantes são praticamente iguais.

Estes resultados não serão de estranhar, uma vez que as tarefas são repetidas sequencialmente, sem outras tarefas a correr concorrentemente nem outras interrupções a serem geradas a instantes aleatórios e também não existem operações condicionais que influenciem o fio de execução do programa. Além disso o trabalho é implementado sobre um DSP e uma das razões que levou a indústria ao desenvolvimento de DSPs foram

exactamente situações deste género onde era necessário implementar uma arquitectura de processamento que evitasse a introdução de *jitter* na execução do *software*. Assim, obtém-se um comportamento bastante próximo do pretendido, isto é, sem grandes oscilações nos tempos de execução.

Pela análise das tabelas, verifica-se ainda quando a quantidade de mensagens a escalonar no *Master* é alterada de seis para oito mensagens, os instantes em que os pedidos de transmissão das mensagens síncronas ficam prontos não se alteram pela mesma proporção que nas mudanças anteriores. Assim foi investigada e apurada a razão deste comportamento verificando-se que o *Master* não conseguia cumprir as *deadlines* impostas, as quais eram iguais ao período.

Foi ainda reduzido o número de mensagens a escalonar para sete, mas o resultado foi semelhante. De seguida foi ainda reduzido o tamanho desta sétima mensagem na tentativa de que o *Master* conseguisse incluí-la no escalonamento para o EC, mas sem sucesso, mesmo com um tamanho de zero b*ytes* a mensagem falha o escalonamento.

Para verificar o comportamento do segundo barramento do DSP foi repetido o processo utilizando os mesmos cenários que no primeiro barramento.

Cenário a) – *Slave* produz uma mensagem síncrona e *Master* escalona duas mensagens – barramento 2.

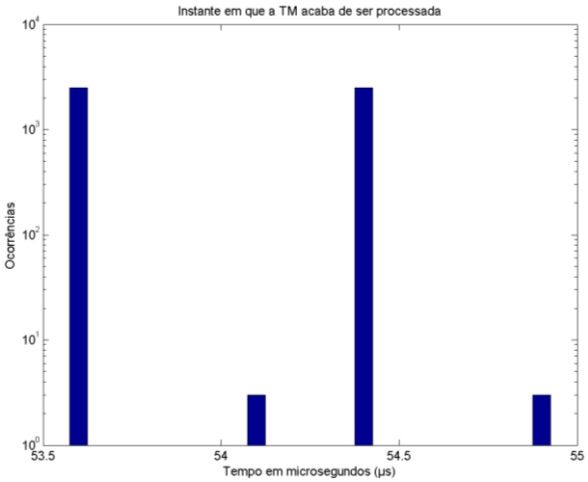

Figura 4.30 – Histograma dos tempos de processamento da TM para o cenário a) – barramento 2.

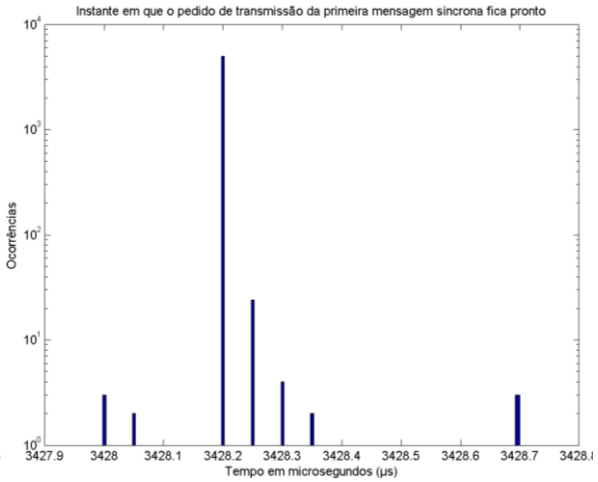

Figura 4.31 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário a) – barramento 2.

A diferença entre o cenário a) do barramento 1 e do barramento 2 é imperceptível, tanto ao nível da latência no processamento da TM, como no instante em

que o pedido de transmissão da primeira mensagem fica pronto, sendo este o resultado que se pretende pois indica que os barramentos estão a ser geridos de forma semelhante.

Cenário b) – *Slave* produz uma mensagem síncrona e *Master* escalona quatro mensagens – barramento 2.

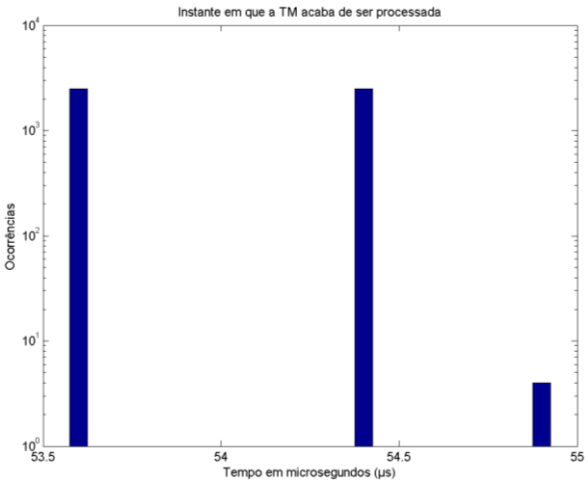

Figura 4.32 - Histograma dos tempos de processamento da TM para o cenário b) – barramento 2.

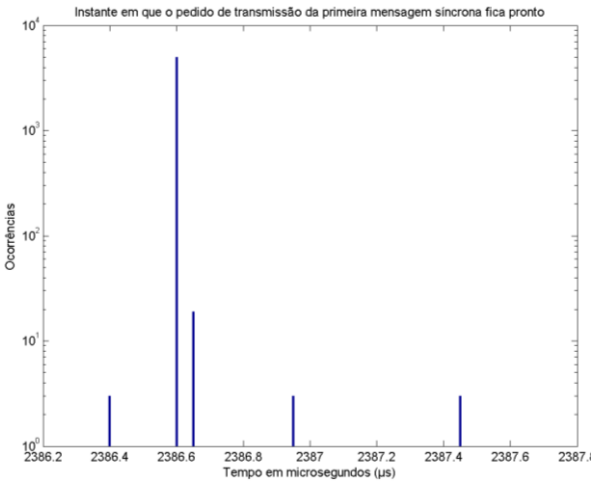

Figura 4.33 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário b) – barramento 2.

Assim como no cenário a), todo este comportamento se assemelha ao cenário homólogo do primeiro barramento, verificando-se mais uma vez a coerência entre a gestão de ambos os barramentos.

Cenário c) – *Slave* produz uma mensagem síncrona e *Master* escalona seis mensagens – barramento 2.

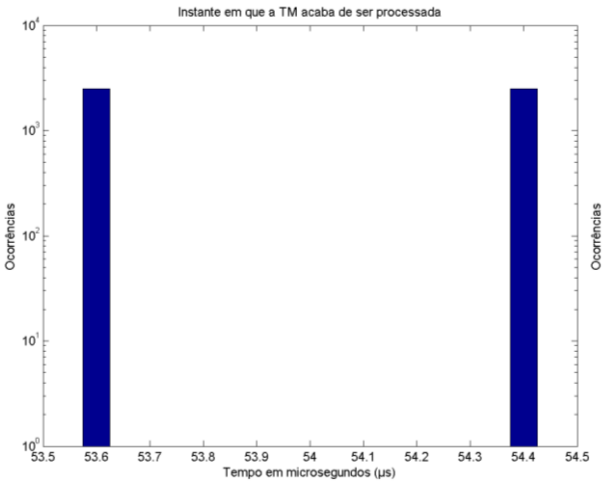

Figura 4.34 - Histograma dos tempos de processamento da TM para o cenário c) – barramento 2.

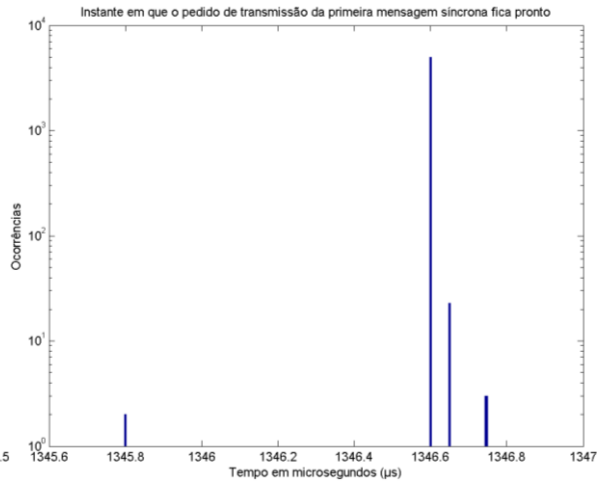

Figura 4.35 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário c) – barramento 2.

Aqui se verifica novamente que os valores estão de acordo com o previsto pelas razões já apresentadas para o mesmo cenário no barramento 1.

Cenário d) – *Slave* produz uma mensagem síncrona e *Master* escalona oito mensagens – barramento 2.

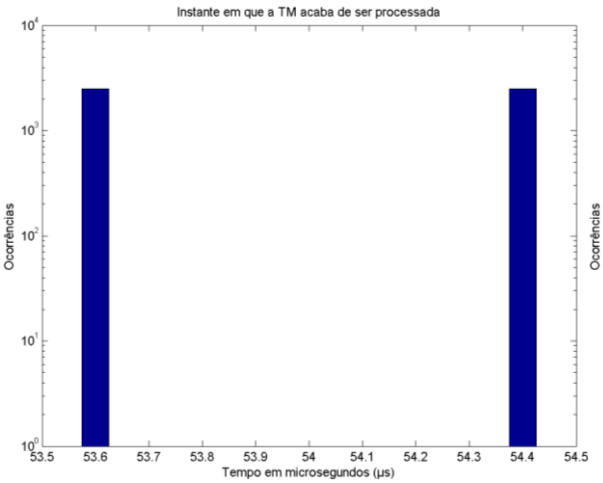

Figura 4.36 - Histograma dos tempos de processamento da TM para o cenário d) – barramento 2.

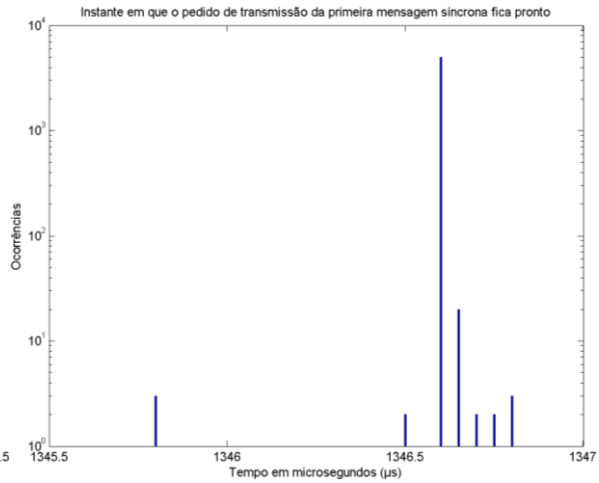

Figura 4.37 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário d) – barramento 2.

De modo semelhante ao cenário d) do barramento 1 o *Master* falha as *deadlines* e o instante de início de transmissão da primeira mensagem síncrona é também próximo de 1,35 ms.

Cenário e) – *Slave* produz duas mensagens síncronas e *Master* escalona duas mensagens – barramento 2.

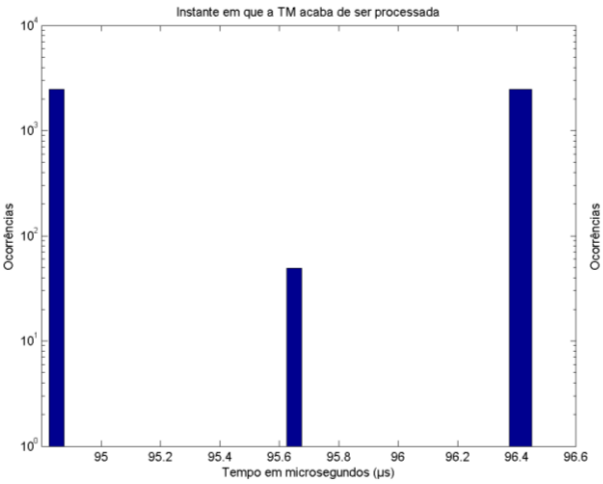

Figura 4.38 - Histograma dos tempos de processamento da TM para o cenário e) – barramento 2.

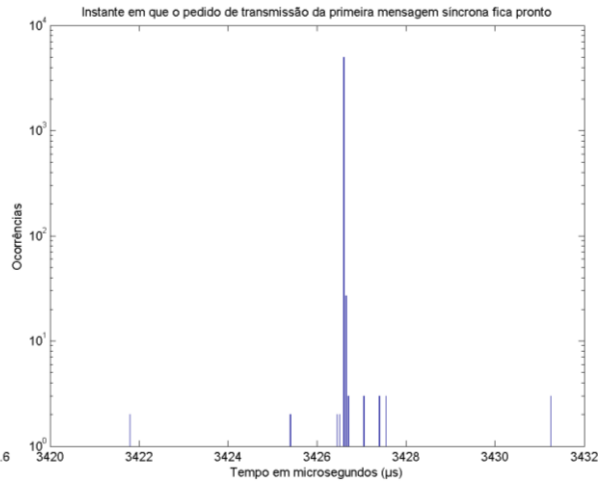

Figura 4.39 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário e) – barramento 2.

Comparando com o mesmo cenário no caso do primeiro barramento, aqui a latência medida sofre o mesmo acréscimo em relação aos cenários anteriores, pelas razões já apresentadas no cenário e) do primeiro barramento.

Cenário f) – *Slave* produz duas mensagens síncronas e *Master* escalona quatro mensagens – barramento 2.

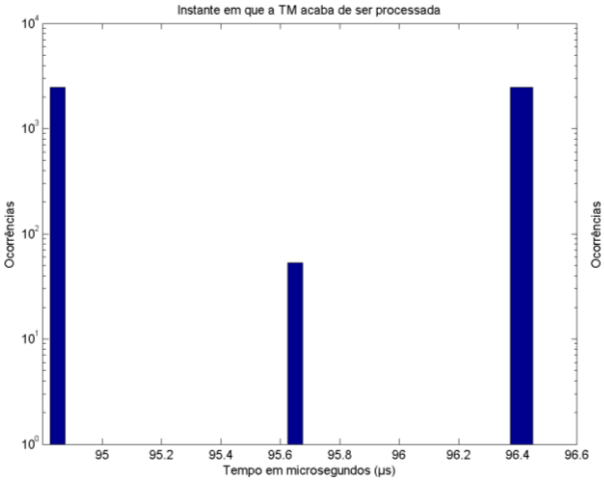

Figura 4.40 - Histograma dos tempos de processamento da TM para o cenário f) – barramento 2.

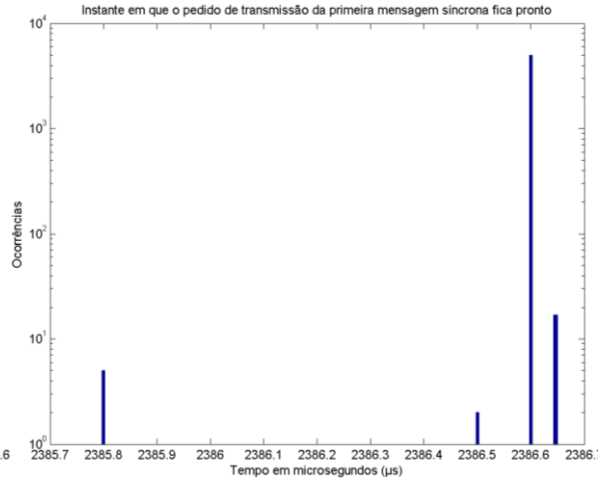

Figura 4.41 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário f) – barramento 2.

As alterações registadas neste cenário são semelhantes às registadas para o cenário homólogo do barramento 1.

Cenário g) – *Slave* produz duas mensagens síncronas e *Master* escalona seis mensagens – barramento 2.

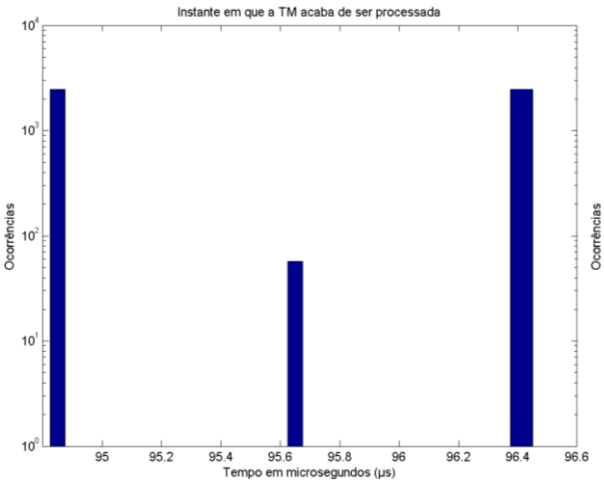

Figura 4.42 - Histograma dos tempos de processamento da TM para o cenário g) – barramento 2.

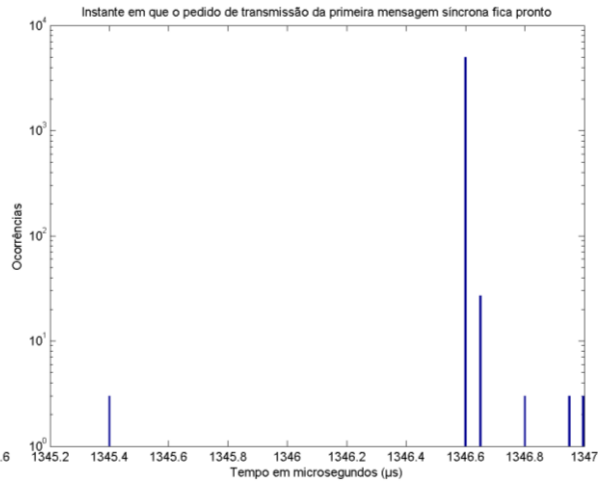

Figura 4.43 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário g) – barramento 2.

Mais uma vez os valores de latência não se alteram pois não foi alterado o número de mensagens a produzir pelo nodo. O instante em que o pedido de transmissão da primeira mensagem síncrona fica pronto também está de acordo com o número de mensagens a escalonar pelo *Master*.

Cenário h) – *Slave* produz duas mensagens síncronas e *Master* escalona oito mensagens – barramento 2.

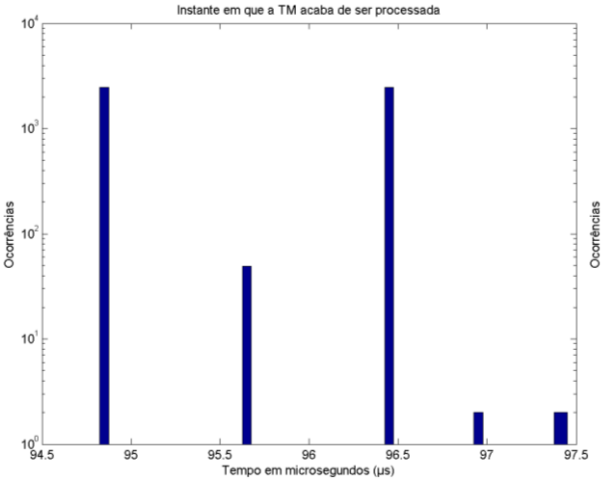

Figura 4.44 - Histograma dos tempos de processamento da TM para o cenário h) – barramento 2.

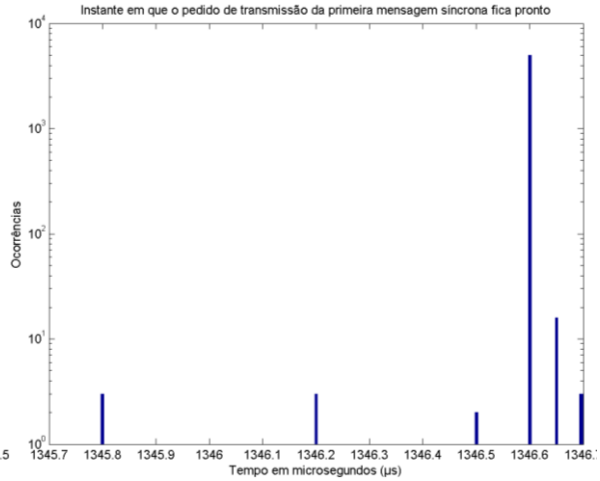

Figura 4.45 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário h) – barramento 2.

As alterações registadas são em tudo semelhantes às registadas no cenário h) do barramento 1.

Cenário i) – *Slave* produz três mensagens síncronas e *Master* escalona quatro mensagens – barramento 2.

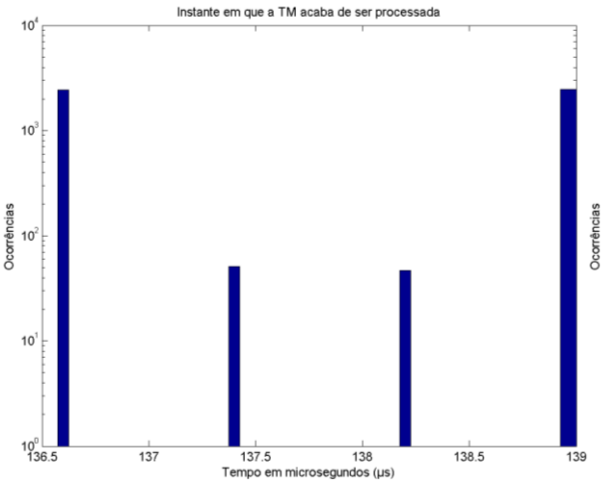

Figura 4.46 - Histograma dos tempos de processamento da TM para o cenário i) – barramento 2.

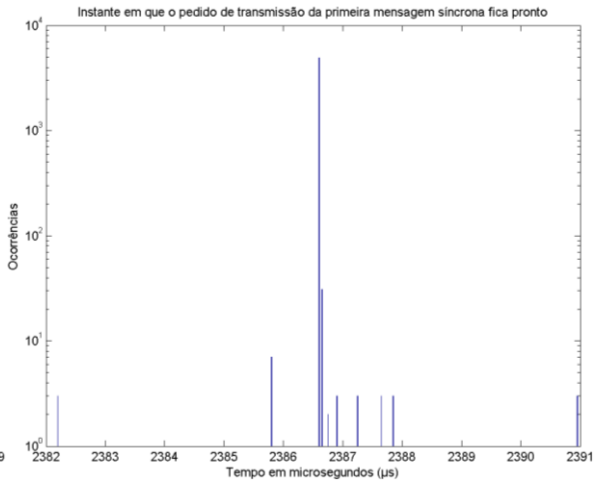

Figura 4.47 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário i) – barramento 2.

O cenário i) mostra que os valores obtidos para a latência introduzida pelo processamento da TM são semelhantes aos do cenário i) do primeiro barramento.

Cenário j) – *Slave* produz três mensagens síncronas e *Master* escalona seis mensagens – barramento 2.

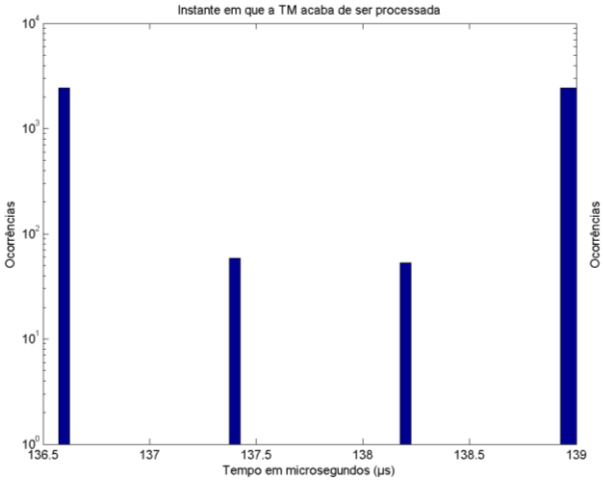

Figura 4.48 - Histograma dos tempos de processamento da TM para o cenário j) – barramento 2.

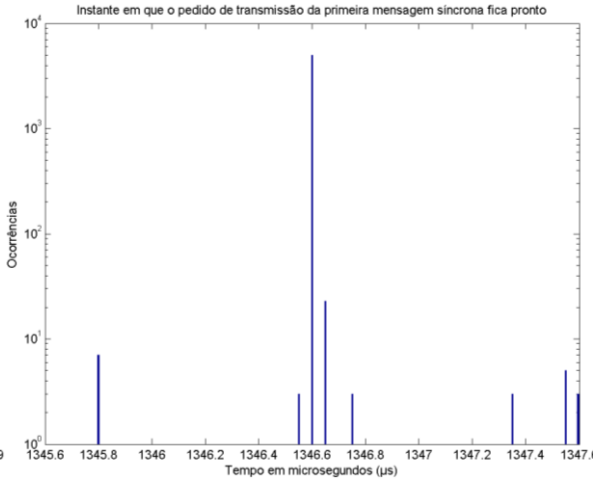

Figura 4.49 - Histograma dos tempos em que o pedido da primeira mensagem síncrona fica pronto no cenário j) – barramento 2.

Mais uma vez os valores obtidos estão de acordo com os obtidos para o primeiro barramento em ambas as situações deste cenário.

Cenário k) – *Slave* produz três mensagens síncronas e *Master* escalona oito mensagens – barramento 2.

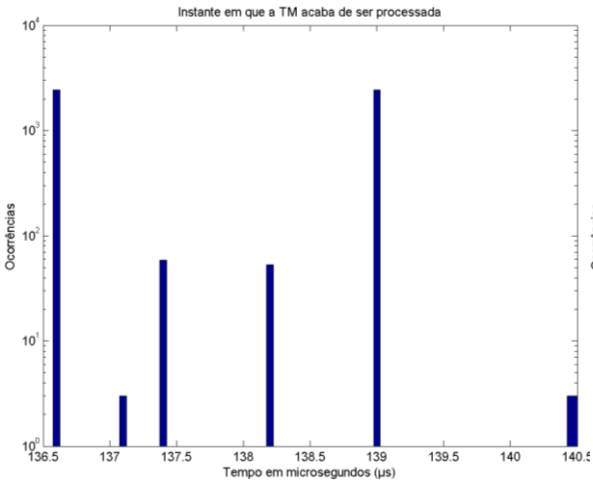

Figura 4.50 - Histograma dos tempos de processamento da TM para o cenário k) – barramento 2.

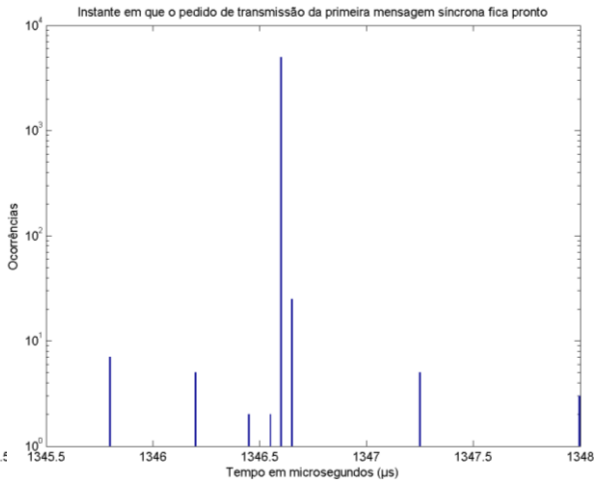

Figura 4.51 - Histograma dos tempos em que o pedido da primeira mensagem síncrona fica pronto no cenário k) – barramento 2.

Em conclusão, da análise dos vários cenários verifica-se que os barramentos 1 e 2 do DSP funcionam da mesma maneira, mantendo latências e instantes de transmissão muito próximos. Isto está de acordo com o que era esperado uma vez que o barramento 2 seria replicado do primeiro. Um resumo dos valores estatísticos relevantes para a análise do segundo barramento do DSP está registado nas tabelas 4.7 e 4.8.

Cenário	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Média	54,0	54,0	54,0	54,0	95,7	95,7	95,7	95,7	137,8	137,8	137,8
Desvio Padrão	0,4	0,4	0,4	0,4	0,8	0,8	0,8	0,8	1,2	1,2	1,2
Máximo	54,4	54,9	54,4	54,4	96,5	96,5	96,5	97,5	139,0	139,0	140,5
Mínimo	53,6	53,6	53,6	53,6	94,9	94,9	95,9	94,9	136,6	136,6	136,6

Tabela 4.7 – Dados estatísticos de cada cenário para tempos de processamento da TM no barramento 2. (valores apresentados em µs)

De forma semelhante ao obtido para o primeiro barramento os dados da tabela 4.7 apresenta como valores aproximados dos tempos de processamento da TM 54 µs, 96 µs e 138 µs, para a produção de 1, 2 e 3 mensagens, respectivamente. Representando um acréscimo do *overhead* introduzido por cada mensagem produzida, cerca de 40 µs. A relação verificada anteriormente para o primeiro barramento mantém-se válida também para o segundo.

Cenário	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Média	3426,6	2386,6	1346,6	1346,6	3426,6	2386,6	1346,6	1346,6	2386,6	1346,6	1346,6
Desvio Padrão	0,016	0,020	0,012	0,017	0,119	0,023	0,027	0,018	0,132	0,046	0,045
Máximo	3426,6	2387,4	1346,8	1346,8	3431,3	2386,6	1347,0	1346,7	2390,9	1347,6	1348,0
Mínimo	3425,8	2386,4	1345,8	1345,8	3421,8	2385,8	1345,4	1345,8	2382,2	1345,8	1345,8

Tabela 4.8 – Dados estatísticos de cada cenário para o instante em que o pedido da primeira mensagem síncrona fica pronto no barramento 2. (valores apresentados em µs)

Mais uma vez se verifica a proximidade entre valores de ambos os barramentos. Este facto leva à conclusão que o sistema funciona de forma praticamente igual nos dois barramentos.

De forma a comparar agora o comportamento do sistema em DSP com o sistema inicial de onde se partiu (o PIC18), serão agora efectuados os mesmos testes para PIC18 para que se possa tirar algumas conclusões.

Cenário a) – *Slave* produz uma mensagem síncrona e *Master* escalona duas mensagens – PIC18.

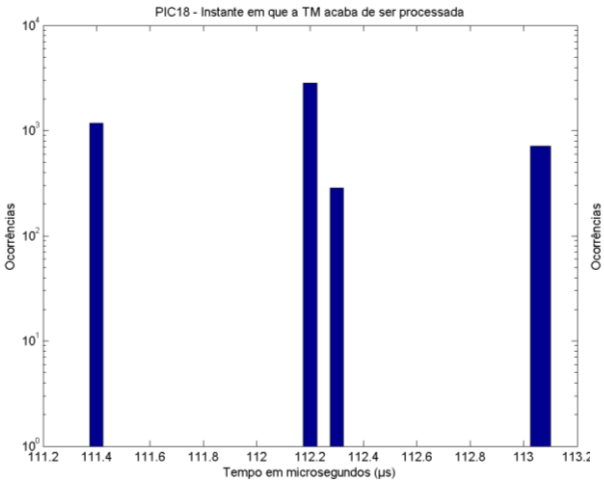

Figura 4.52 - Histograma dos tempos de processamento da TM para o cenário a) – PIC18.

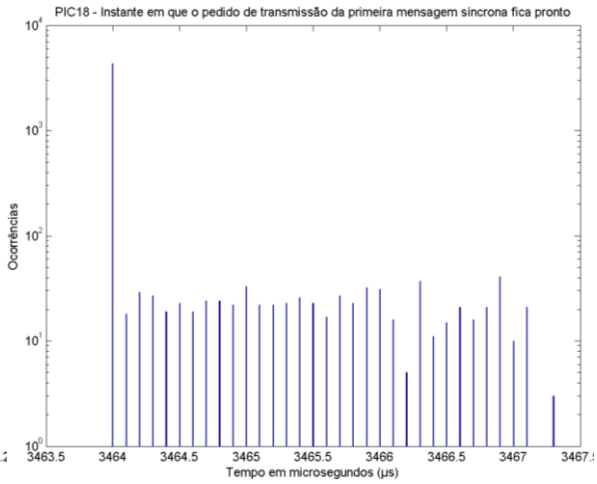

Figura 4.53 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário a) – PIC18.

Comparando o cenário a) do PIC18 com o mesmo cenário de qualquer um dos barramentos do DSP verificam-se imediatamente diferenças óbvias. A latência é aproximadamente o dobro da obtida no DSP, o *jitter* que afecta os valores a medir, quer no terminar de processamento da TM como no instante em que o pedido de transmissão da primeira mensagem síncrona fica pronto, é claramente superior. O instante em que a primeira mensagem síncrona fica pronta a enviar é semelhante ao do DSP, havendo uma diferença de cerca de 40 µs, que representa aproximadamente a diferença entre a carga adicional devido ao processamento da TM, assim uma vez que o DSP é mais rápido, este termina o processamento mais cedo, mas considera na mesma 100 µs, pois eram necessários testes deste género para averiguar o valor real da carga computacional. Então a janela síncrona é iniciada mais cedo assim como a transmissão das mensagens síncronas.

Cenário b) – *Slave* produz uma mensagem síncrona e *Master* escalona quatro mensagens – PIC18.

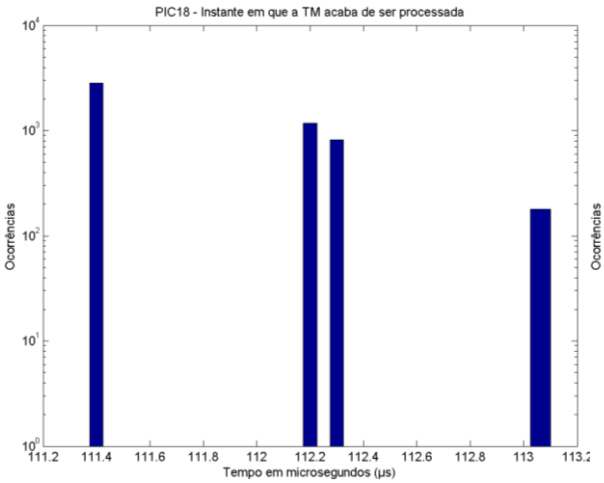

Figura 4.54 - Histograma dos tempos de processamento da TM para o cenário b) – PIC18.

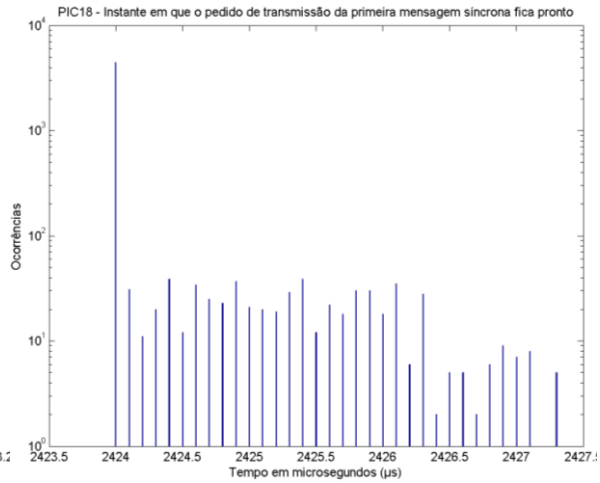

Figura 4.55 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário b) – PIC18.

Os dados aqui obtidos mostram novamente as diferenças em termos de *jitter*, uma vez que os valores oscilam muito mais em torno do valor esperado verifica-se que o *jitter* é bastante superior à versão em DSP, em ambos os casos a medir no cenário b). De modo semelhante ao que acontece no cenário anterior, os valores obtidos para o instante em que o pedido de transmissão da primeira mensagem fica pronto diferem de cerca de 40 µs da versão em DSP devido à carga considerada.

Cenário c) – *Slave* produz uma mensagem síncrona e *Master* escalona seis mensagens – PIC18.

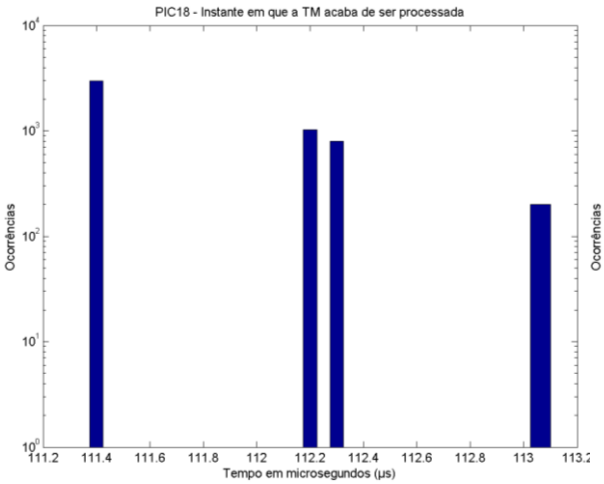

Figura 4.56 - Histograma dos tempos de processamento da TM para o cenário c) – PIC18.

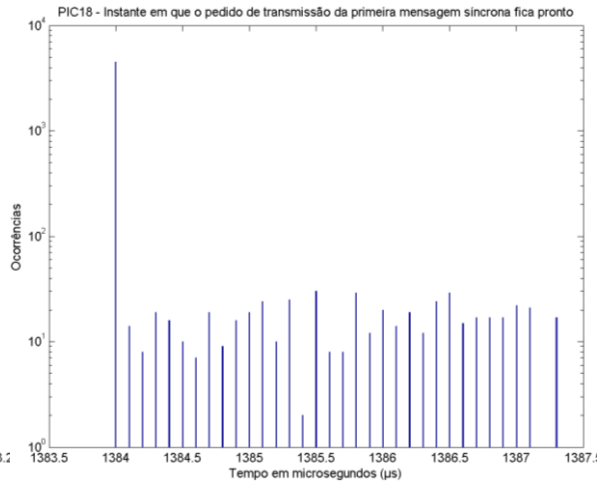

Figura 4.57 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário c) – PIC18.

Mais uma vez a grande diferença a salientar é o *jitter* de que os valores vêm afectados, que é claramente superior à versão em DSP.

Cenário d) – *Slave* produz uma mensagem síncrona e *Master* escalona oito mensagens – PIC18.

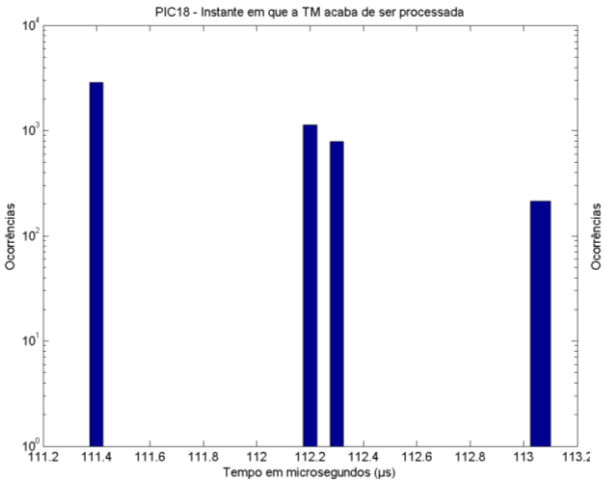

Figura 4.58 - Histograma dos tempos de processamento da TM para o cenário d) – PIC18.

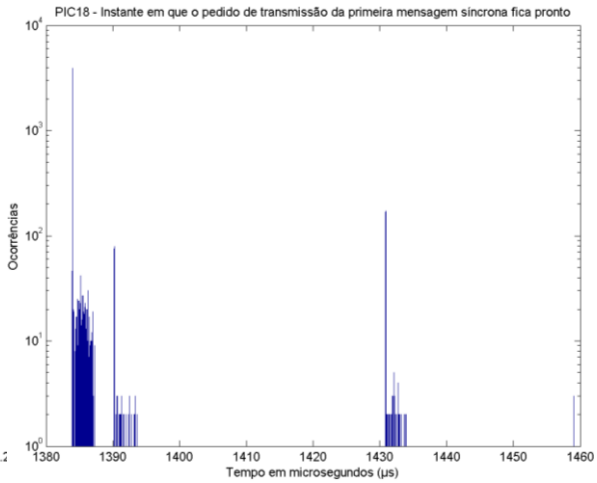

Figura 4.59 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário d) – PIC18.

Neste cenário os valores de *jitter* são ainda maiores que nos cenários anteriores. Esta situação pode estar relacionada com o facto de os *Slaves* pararem de funcionar após enviarem algumas mensagens quando o *Master* falha deadlines. As amostras foram obtidas iniciando os *Slaves* sucessivamente até atingir o número de amostras pretendido. Uma vez que no arranque da execução do PIC os valores são mais instáveis, isso pode explicar a razão de variarem tanto.

Cenário e) – *Slave* produz duas mensagens síncronas e *Master* escalona duas mensagens – PIC18.

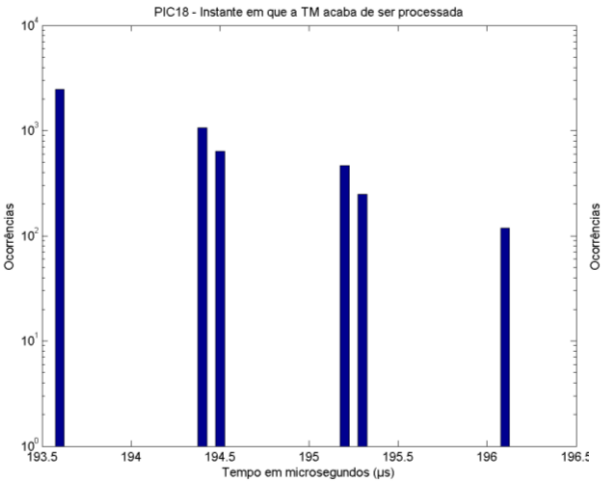

Figura 4.60 - Histograma dos tempos de processamento da TM para o cenário e) – PIC18.

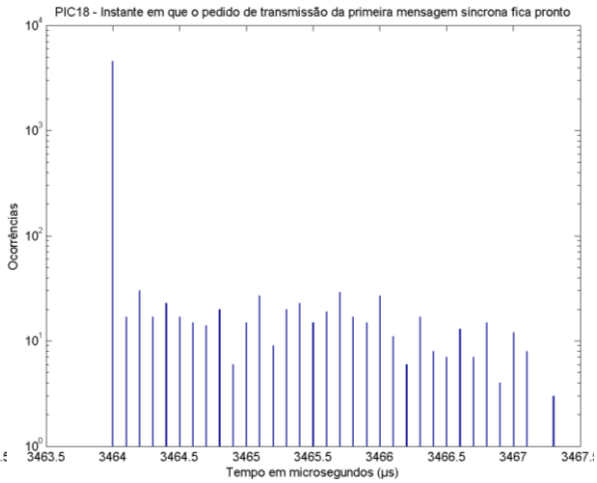

Figura 4.61 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário e) – PIC18.

Neste caso a latência introduzida pelo processamento da TM ultrapassa já o valor máximo obtido para o DSP, isto é utilizando três mensagens e verifica-se um aumento de aproximadamente 100% neste cenário face ao DSP. Além disso, analogamente aos cenários anteriores o *jitter* que afecta os valores continua bastante superior ao do DSP.

Cenário f) – *Slave* produz duas mensagens síncronas e *Master* escalona quatro mensagens – PIC18.

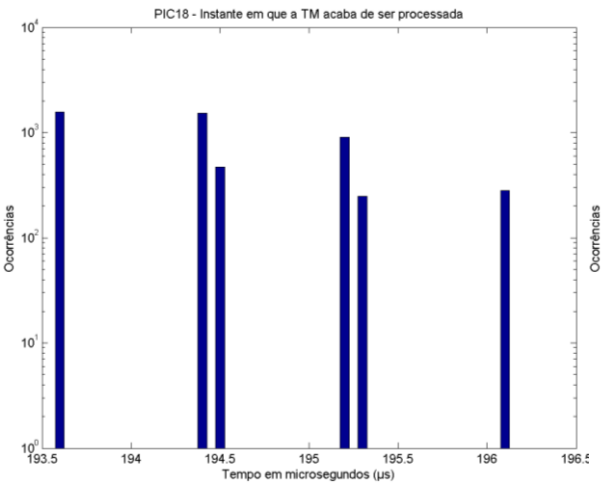

Figura 4.62 - Histograma dos tempos de processamento da TM para o cenário f) – PIC18.

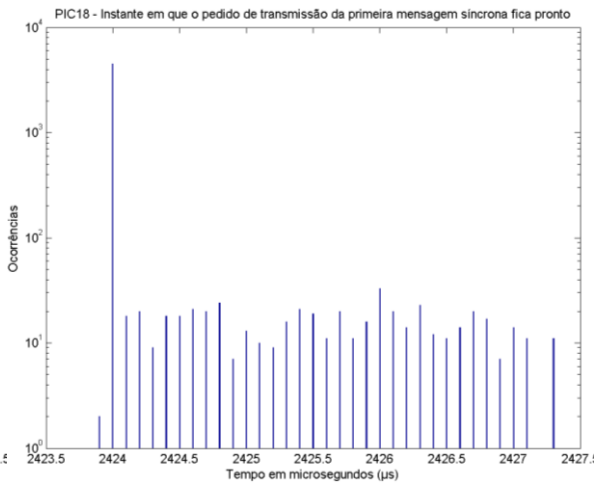

Figura 4.63 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário f) – PIC18.

Os valores aqui obtidos verificam novamente um *jitter* superior e uma latência também bastante mais elevada quando comparados com os valores da versão em DSP. O início de transmissão da primeira mensagem síncrona possui a mesma diferença de 40 µs referida anteriormente.

Cenário g) – *Slave* produz duas mensagens síncronas e *Master* escalona seis mensagens – PIC18.

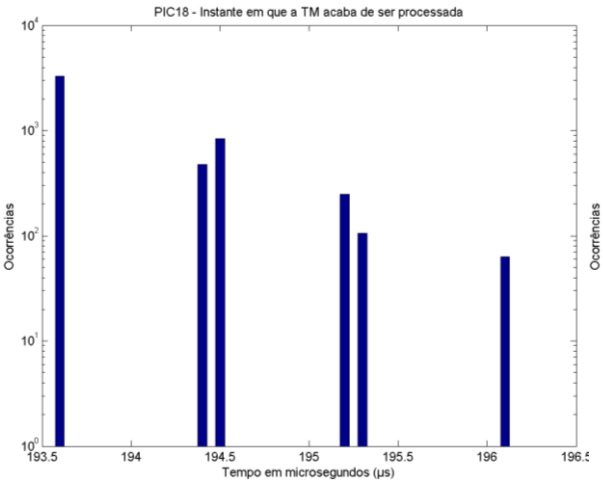

Figura 4.64 - Histograma dos tempos de processamento da TM para o cenário g) – PIC18.

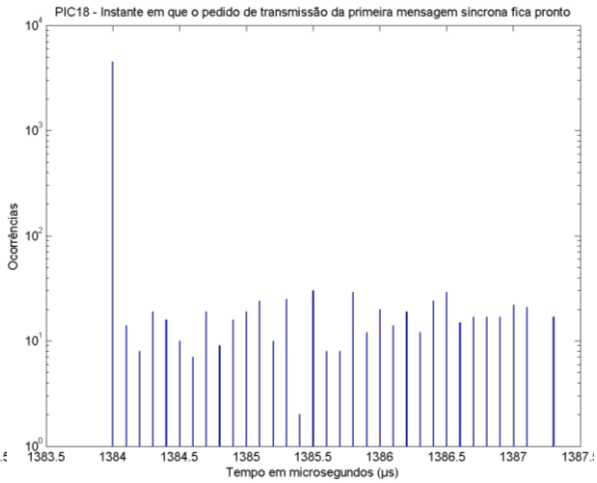

Figura 4.65 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário g) – PIC18.

Novamente se verificam os pontos referidos anteriormente em termos de *jitter* e latência.

Cenário h) – *Slave* produz duas mensagens síncronas e *Master* escalona oito mensagens – PIC18.

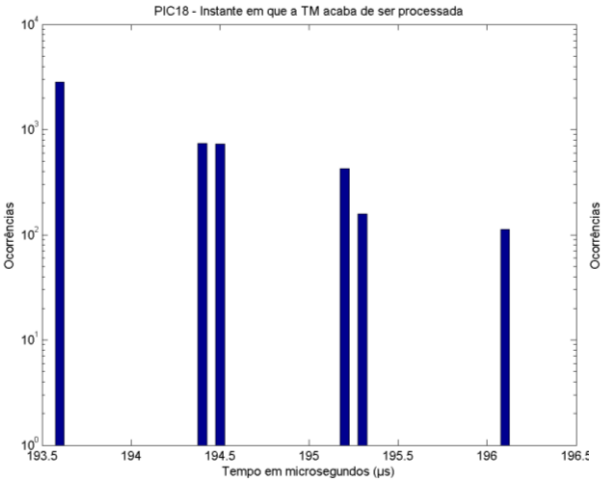

Figura 4.66 - Histograma dos tempos de processamento da TM para o cenário h) – PIC18.

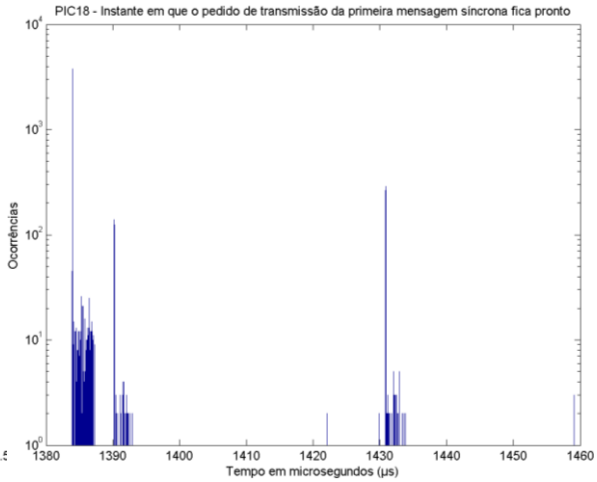

Figura 4.67 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário h) – PIC18.

As questões abordadas anteriormente relativamente a *jitter* e latência verificam-se novamente neste cenário. Mais uma vez ao escalonar oito mensagens o sistema encontra problemas e os *Slaves* não conseguem proceder a um envio contínuo de mensagens síncronas, pelo que os valores obtidos variam mais que os anteriores.

Cenário i) – *Slave* produz três mensagens síncronas e *Master* escalona quatro mensagens – PIC18.

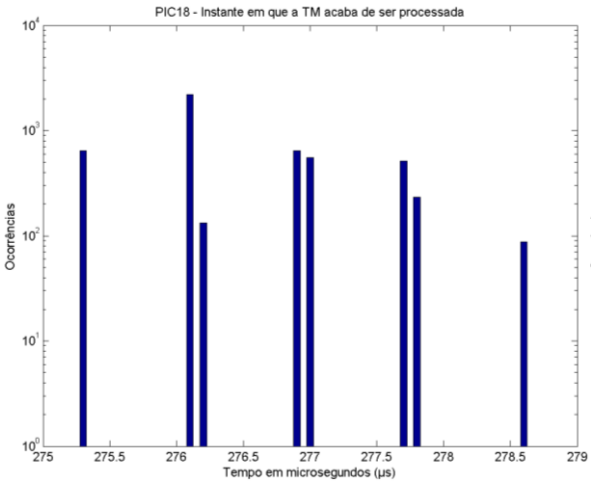

Figura 4.68 - Histograma dos tempos de processamento da TM para o cenário i) – PIC18.

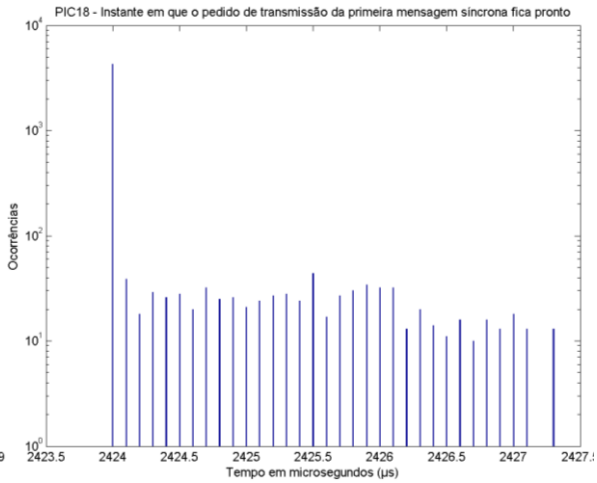

Figura 4.69 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário i) – PIC18.

Neste cenário verifica-se já uma latência rondando os 277 µs, o que representa novamente um aumento de cerca de 100% face ao DSP, o que faz sentido uma vez que a velocidade de execução do DSP é o dobro da velocidade do PIC. Os valores de *jitter* são mais uma vez bastante elevados.

Cenário j) – *Slave* produz três mensagens síncronas e *Master* escalona seis mensagens – PIC18.

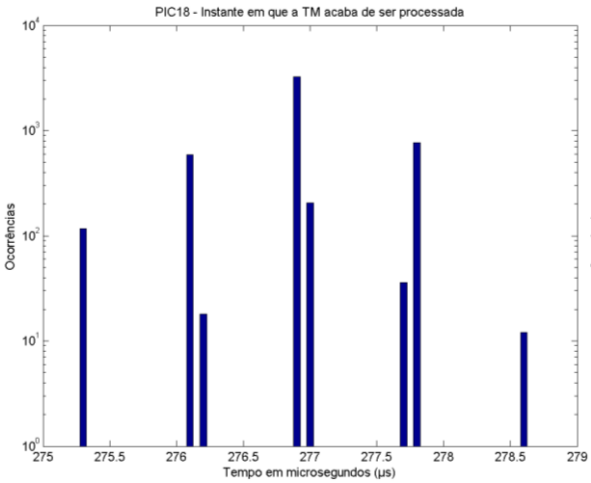

Figura 4.70 - Histograma dos tempos de processamento da TM para o cenário j) – PIC18.

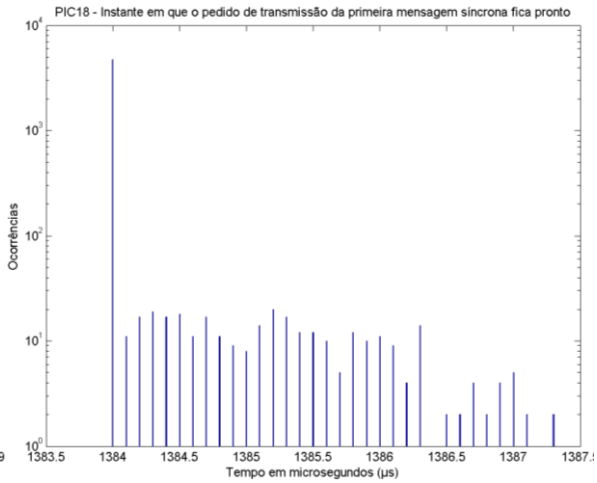

Figura 4.71 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário j) – PIC18.

Uma vez que a latência, introduzida pelo descodificar da TM, está intimamente ligada ao número de mensagens a produzir pelo nodo, esta é semelhante ao cenário anterior uma vez que é produzido o mesmo número de mensagens síncronas. O *jitter* é mais uma vez elevado.

Cenário k) – *Slave* produz três mensagens síncronas e *Master* escalona duas mensagens – PIC18.

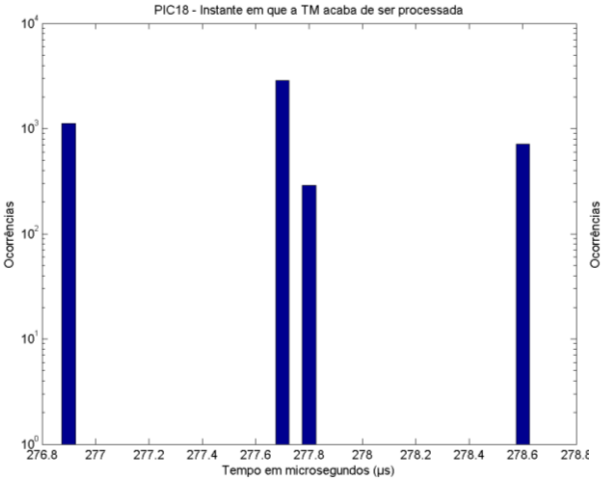

Figura 4.72 - Histograma dos tempos de processamento da TM para o cenário k) – PIC18.

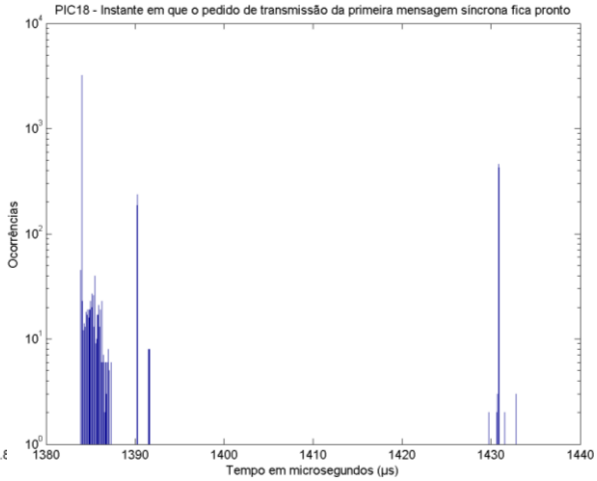

Figura 4.73 - Histograma dos tempos em que o pedido de transmissão da primeira mensagem síncrona fica pronto no cenário k) – PIC18.

Mais uma vez os *Slaves* encontram problemas ao enviar as mensagens síncronas e o *jitter* obtido para o instante em que o pedido de transmissão das mensagens síncronas fica pronto é novamente bastante elevado. As tabelas 4.9 e 4.10 apresentam um resumo dos valores estatísticos para comparação com os valores registados anteriormente.

Cenário	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Média	112,1	111,8	111,8	111,8	194,2	194,4	194,0	194,1	276,5	277,0	277,7
Desvio Padrão	0,5	0,5	0,5	0,5	0,7	0,7	0,6	0,6	0,8	0,5	0,5
Máximo	113,1	113,1	113,1	113,1	196,1	196,1	196,1	196,1	278,6	278,6	278,6
Mínimo	111,4	111,4	111,4	111,4	193,6	193,6	193,6	193,6	275,3	275,3	276,9

Tabela 4.9 – Dados estatísticos de cada cenário para tempos de processamento da TM no PIC18. (valores apresentados em µs)

A tabela 4.7 apresenta como valores aproximados dos tempos de processamento da TM 112 µs, 194 µs e 277 µs, para a produção de 1, 2 e 3 mensagens, respectivamente. Representando um acréscimo do *overhead* introduzido por cada mensagem produzida, cerca de 85 µs.

Cenário	a)	b)	c)	d)	e)	f)	g)	h)	i)	j)	k)
Média	3464,2	2424,2	1384,2	1387,8	3464,1	2424,2	1384,2	1390,0	2424,2	1384,1	1393,0
Desvio Padrão	0,6	0,5	0,6	12,2	0,5	0,5	0,6	15,0	0,6	0,3	17,7
Máximo	3467,3	2427,3	1387,3	1459,0	3467,3	2427,3	1387,3	1459,1	2427,3	1387,3	1432,8
Mínimo	3464,0	2424,0	1383,9	1383,9	3464,0	2423,9	1384,0	1383,9	2424,0	1384,0	1383,9

Tabela 4.10 – Dados estatísticos de cada cenário para o instante em que o pedido da primeira mensagem síncrona fica pronto no PIC18. (valores apresentados em µs)

A tabela 4.10 resume os valores médios, máximos, mínimos e desvio padrão obtidos ao longo dos ensaios. Verifica-se que existem alguns valores que não saem um pouco do esperado, como o caso dos cenários d), h) e k), o que se deve à forma como as amostras foram retiradas uma vez que os *Slaves* encontravam problemas ao tentar enviar as mensagens síncronas.

As medições feitas permitem verificar que o sistema em DSP se comporta da mesma forma que o sistema em PIC18, mas com importantes melhorias. Verifica-se que o sistema permite realmente dobrar a largura de banda existente no PIC18, foram ganhos cerca de 40 µs, 100 µs e 140 µs para utilização no envio de uma, duas e três mensagens, respectivamente, o que utilizando taxas de transmissão superiores pode representar uma melhoria bastante significativa. Após a análise destes resultados foi ajustado o valor da carga considerada para processamento da TM.

4.1.5. Verificação do sistema de mensagens assíncronas

A implementação do protocolo utilizando dois barramentos obrigou à introdução de código e modificação de código existente. Por esta razão deve estar garantido que as alterações de código introduzidas não tenham afectado o sistema de mensagens assíncronas. Para realizar esta análise, o procedimento utilizado consistiu em enviar uma mensagem assíncrona do *Slave* 1 para o *Slave* 2 por cada EC e verificar a sua presença no barramento e recepção no nodo de consumo. A taxa de transmissão utilizada no barramento foi novamente de 250kbps. A figura 4.30 ilustra o sistema utilizado para esta experiência.

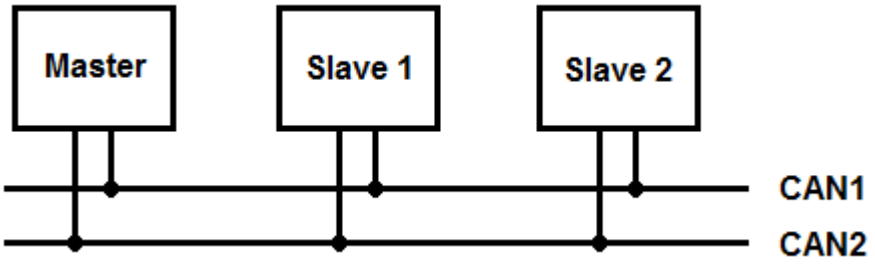

Figura 4.30 – Esquema da montagem utilizada na verificação do subsistema de mensagens assíncronas.

De modo semelhante ao procedimento utilizado para testar o sistema de mensagens síncronas, foi utilizada também uma técnica de *sequence number* para que cada mensagem tenha uma identificação única. Deste modo foi aproveitado o código desenvolvido anteriormente e utilizada uma variável de 8 *bits* que por cada EC, é incrementada e o conteúdo da mensagem preenchido com {X,X+1,…,X+7} e realiza o seu envio. Deve ser tido em consideração ainda que o sistema de mensagens assíncronas apenas está implementado num dos barramentos, o CAN1, por onde as mensagens seguem obrigatoriamente. Os resultados obtidos são apresentados nas figuras 4.31.

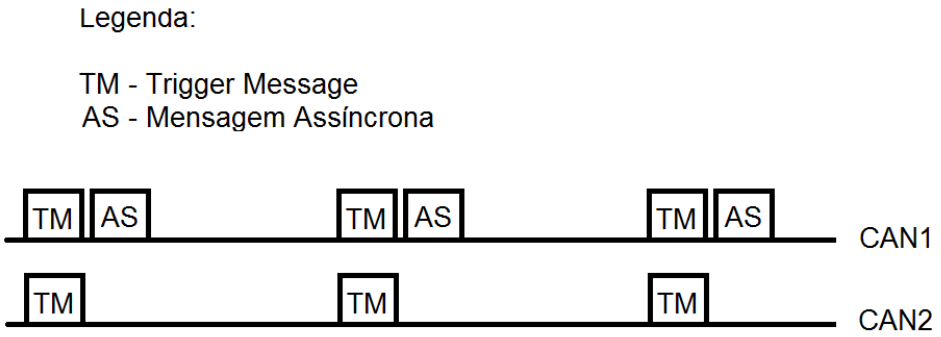

Figura 4.31 – Visualização das mensagens no barramento.

Dos resultados obtidos, verifica-se que o envio está de facto a ocorrer e ainda que a escrita e leitura nos *buffers* CAN está a ser realizada de forma correcta, o que significa que o sistema de mensagens assíncronas não foi afectado pelas alterações introduzidas na implementação em DSP.

4.1.6. Compatibilidade com FTT-CAN baseado no PIC18F258

Uma vez que o formato da trama CAN nada tem a ver com a plataforma, o que se espera da implementação em DSP é que o envio seja feito de forma transparente mantendo a compatibilidade a versão em PIC. Para garantir que as alterações no código FTT não influenciaram esta compatibilidade entre sistemas foi realizada uma verificação de compatibilidade com o sistema antigo utilizado um PIC18F258 correndo o FTT-CAN como consumidor reportando as mensagens CAN recebidas através da porta série. Como produtor foi utilizado um DSP, produzindo uma mensagem crítica (enviada por ambos os barramentos), mensagens não críticas por cada barramento e mensagem assíncrona pelo barramento em que está implementado. Na figura 4.32 pode ser consultado o esquema da montagem utilizada.

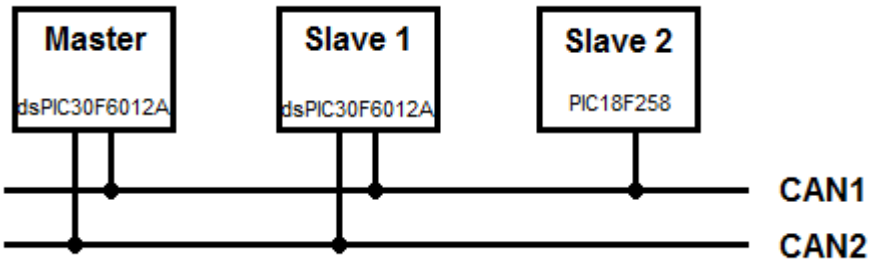

Figura 4.32 – Esquema da montagem utilizada no teste de compatibilidade.

Dando inicio à transmissão, os resultados obtidos foram estes:

```
Synch: Critsyn
Synch: Bus1syn
Assync: CritAsy
Synch: Critsyn
Synch: Bus1syn
Assync: CritAsy
```

```
Synch: CritSyn
Synch: Bus2Syn
Synch: CritSyn
Synch: Bus2Syn
Synch: CritSyn
Synch: Bus2Syn
```

Figura 4.33 – Mensagens recebidas no PIC18.

No lado esquerdo da figura 4.33 estão registadas as mensagens recebidas no PIC18 ao ser conectado ao CAN1, à direita encontra-se o registo das mensagens recebidas quando o PIC18 é conectado ao CAN2, verificando-se uma correspondência exacta face às mensagens enviadas pelo produtor implementado no DSP. Verificando-se assim que os

sistemas são compatíveis e funcionam de forma correcta quando interligados e configurados correctamente.

Capítulo 5

Conclusões e trabalho futuro

5.1. Conclusões

Do trabalho desenvolvido e testes realizados é possível verificar que a implementação do protocolo na sua versão para barramento replicado funciona de modo correcto. O envio e a recepção da informação estão a ser realizados de forma correcta em ambos os barramentos. A descodificação da TM encontra-se a ser processada de forma correcta e os *Slaves* comportam-se de acordo com a informação codificada na TM.

Da análise dos tempos de processamento, verifica-se que a descodificação da TM apresenta um *overhead* de computacional de cerca de 55 µs para a produção de uma mensagem no *Slave*, sendo acrescido de cerca de 40 µs por cada mensagem adicional que o nodo produza. Os instantes a que os pedidos de transmissão das mensagens síncronas ficam prontos levaram à conclusão de que o *jitter* introduzido é baixo e muito menor que na versão em PIC18, além disso o sistema suporta 6 mensagens de tamanho máximo por EC, quando este tem duração 5 ms, uma vez que o *Master* acusa falha de *deadline* com mais mensagens de tamanho máximo.

Os testes realizados indicam ainda que a transmissão e recepção de mensagens assíncronas não foram afectadas negativamente com as alterações introduzidas e funcionam correctamente no barramento em que o subsistema está implementado.

Verificou-se ainda que os *Slaves* do FTT-CAN na implementação do protocolo em DSP apresentam são compatíveis com os *Slaves* do FTT-CAN implementados em PIC18F258, como seria de esperar, podendo coexistir no mesmo sistema.

Contudo, o mecanismo de gestão de redundância, da responsabilidade do *Master*, não estaria a efectuar a mudança de barramento e ao averiguar a causa desta situação, verificou-se que o código responsável por esta gestão estaria comentado não sendo

considerado na compilação. Ao incluir o código na compilação a mudança de barramentos efectuou-se de forma correcta concluindo-se que o sistema de redundância funciona de forma correcta.

5.2. Trabalho futuro

Existe ainda algum trabalho a realizar para dar como terminada a implementação do protocolo FTT-CAN utilizando barramento replicado. As duas tarefas mais prioritárias tendo em conta o estado actual do trabalho seriam, em primeiro lugar a implementação do subsistema de mensagens assíncronas em barramento duplo, a segunda tarefa seria dedicada a implementar o subsistema de mensagens de controlo também em barramento replicado. Posteriormente, podia ser avaliada a implementação de todo o subsistema de *gateway*, pois foi algo que nem sequer foi ponderado, mas que faz parte da implementação do FTT-CAN em PIC18F258.

Existe ainda algum potencial para a optimização do código desenvolvido, como referido em pontos anteriores desta Dissertação, o que pode trazer melhorias no desempenho global do sistema.

Seria também interessante ponderar mais tarde a possibilidade de replicar os próprios nodos *Slave*, à semelhança do que acontece no protocolo TTP/C e com o *Master* do próprio FTT-CAN. Deste modo seria conseguida redundância ao nível de todo o hardware e teríamos um mecanismo que iria permitir contornar avarias dos nodos que se encontrassem replicados.

Para utilização deste protocolo em aplicações que requeiram uso de bateria, poderá ainda ser importante reduzir a potência consumida por cada placa, recorrendo-se para isso à diminuição da tensão de alimentação dos componentes, uma vez que estas possuem um elevado consumo tendo um impacto bastante significativo na autonomia do sistema. Além disso, os DSPs utilizados permitem baixar a tensão até 2,5V. Mas a redução de tensão implica uma redução da frequência de operação e logo uma degradação do desempenho, pelo que será necessário avaliar o seu impacto no desempenho do sistema.

Capítulo 6

Referências

[1] FlexRay, "FlexRay Communications System Protocol Specification", version 2.1, 2005, revision A, disponível em: <www.iestcfa.org/presentations/.../WFCS2006_FlexRay_v2b-1.pdf>, acedido em: 25-09-2010.

[2] Kopetz, H., Grunsteidl, G., "TTP-A protocol for Fault-Tolerant Real-Time Systems", computer, vol. 27, issue 1, Jan 1994, pp 14-23.

[3] González, M., "Improving Error Containment and Reliability of Communication Subsystems Based on Controller Area Network (CAN) by Means of Adequate Star Topologies", PhD Thesis, Universitat de Les Illes Balears

[4] Silva, V., "Flexible Redundancy and Bandwidth Management in Fieldbuses", PhD Thesis, Universidade de Aveiro

[5] Silva, V., Marau, R.,Almeida, L., Ferreira, J., Calha, M., Pedreiras, P., Fonseca, J., "Implementing a distributed sensing and actuation system: The CAMBADA robots case study", in Proc. ETFA 2005, September 2005, pp. 781-788.

[6] Silva, V., Fonseca, J., "Using FTT-CAN to combine redundancy with increased bandwidth", in Proc. WFCS 2006, September 2006, pp. 54-62

[7] Introdution to CAN controllers at ST, disponível em: <http://www.datasheetarchive.com/pdf/getfile.php?dir=Datasheet-072&file=DSA00363432.pdf&scan=n >, acedido em: 20-09-2010.

[8] Pimentel IECON'01: The 27th Annual Conference of the IEEE Industrial Electronics Society, in Proc. IECON 2001, 2001, pp. 1800 – 1805 vol. 3

[9] Fujitsu Microelectronics (Shanghai) Co.,Ltd., "Next Generation Car Network – Flexray, Fujitsu", Jun 2006, disponivel em: < http://www.fujitsu.com/downloads/CN/fmc/lsi/FlexRay-EN.pdf >, acedido em: 30-09-2010.

[10] - Mr. Steve Talbot, Dr. Shangping Ren, Illinois Institute of Technology, "Deeply Embedded Control Systems, Cyber Physical Systems (CPS) in Passenger Vehicles: Survey of CAN, TTCAN, FlexRay, LIN", disponível em: <http://www.talbotsystems.com/documents/CS_521_Presentation_Talbot.pdf>, acedido em: 31-09-2010.